Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	√	√	√	√	*	*
CCR							

* V Always cleared.

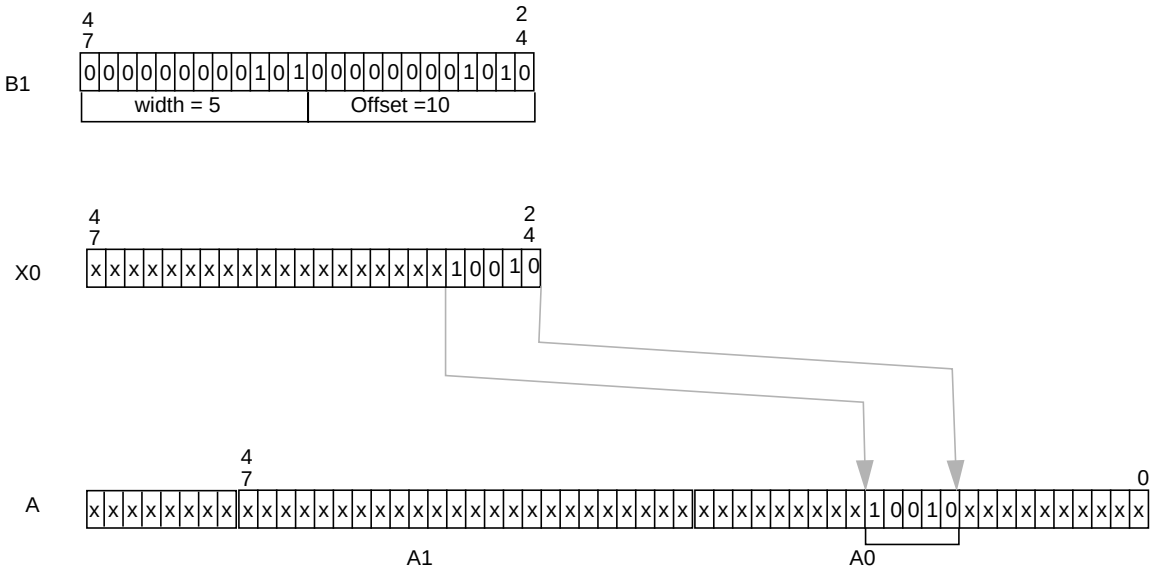
* C Always cleared.

— Unchanged by the instruction.

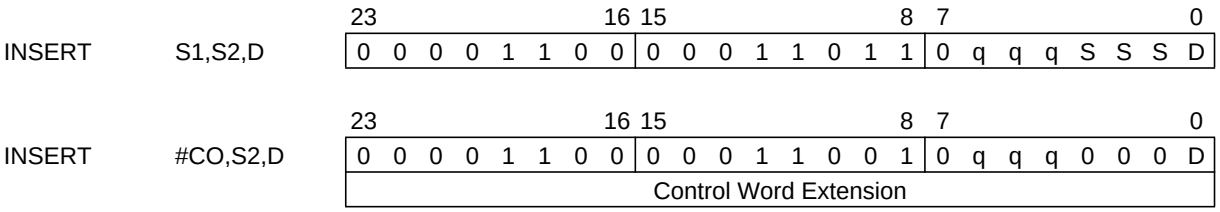
√ Changed according to the standard definition.

Example

INSERT B1,X0,A



Instruction Formats and opcodes



Operation

If cc, then 0xxx → PC
else PC + 1 → PC

If cc, then ea → PC
else PC + 1 → PC

Assembler Syntax

Jcc xxx

Jcc ea

Instruction Fields

{cc}	**CCCC**	Condition code (see **Table 12-18** on page 12-28)
{xxx}	**aaaaaaaaaaaa**	Short Jump Address
{ea}	**MMMRRR**	Effective Address (see **Table 12-13** on page 12-22)

Description Jump to the location in program memory given by the instruction's effective address if the specified condition is true. If the specified condition is false, the Program Counter (PC) is incremented and the effective address is ignored. However, the address register specified in the effective address field is always updated independently of the specified condition. All memory-alterable addressing modes can be used for the effective address. A Fast Short Jump addressing mode can also be used. The 12-bit data is zero-extended to form the effective address. The conditions specified by "cc" are listed on **Table 12-18** on page 12-28.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

Jcc xxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	C	C	C	C	a	a	a	a	a	a	a	a	a	a	a	a

Jcc ea

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	1	M	M	M	R	R	R	1	0	1	0	C	C	C	C
Optional Effective Address Extension																							

JCLR — Jump if Bit Clear — JCLR

Operation

If	S{n} = 0	then	xxxx	→	PC
		else	PC + 1	→	PC
If	S{n} = 0	then	xxxx	→	PC
		else	PC + 1	→	PC
If	S{n} = 0	then	xxxx	→	PC
		else	PC + 1	→	PC
If	S{n} = 0	then	xxxx	→	PC
		else	PC + 1	→	PC
If	S{n} = 0	then	xxxx	→	PC
		else	PC + 1	→	PC

Assembler Syntax

JCLR	#n,[X or Y]:ea,xxxx
JCLR	#n,[X or Y],aa,xxxx
JCLR	#n,[X or Y]:pp,xxxx
JCLR	#n,[X or Y]:qq,xxxx
JCLR	#n,S,xxxx

Instruction Fields

{#n}	**bbbb**	Bit number [0–23]
{ea}	**MMMRRR**	Effective Address (see **Table 12-13** on page 12-22)
{X/Y}	**S**	Memory Space [X,Y] (see **Table 12-13** on page 12-22)
{xxxx}		24-bit absolute Address extension word
{aa}	**aaaaaa**	Absolute Address [0–63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0 – $FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80 – $FFFFBF]
{S}	**DDDDDD**	Source register [all on-chip registers] (see **Table 12-13** on page 12-22)

Description Jump to the 24-bit absolute address in program memory specified in the instruction's 24-bit extension word if the n^{th} bit of the source operand S is clear. The bit to be tested is selected by an immediate bit number from 0–23. If the specified memory bit is not clear, the Program Counter (PC) is incremented and the absolute address in the extension word is ignored. However, the address register specified in the effective address field is always updated independently of the state of the n^{th} bit. All address register indirect addressing modes can reference the source operand S. Absolute Short and I/O Short addressing modes can also be used.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and opcodes

JCLR #n,[X or Y]:ea,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	1	M	M	M	R	R	R	1	S	0	0	b	b	b	b
Absolute Address Extension																							

JCLR #n,[X or Y]:aa,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	a	a	a	a	a	a	1	S	0	0	b	b	b	b
Absolute Address Extension																							

JCLR #n,[X or Y]:pp,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	0	p	p	p	p	p	p	1	S	0	0	b	b	b	b
Absolute Address Extension																							

JCLR #n,[X or Y]:qq,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	0	q	q	q	q	q	q	1	S	0	0	b	b	b	b
Absolute Address Extension																							

JCLR #n,S,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	1	D	D	D	D	D	D	0	0	0	0	b	b	b	b
Absolute Address Extension																							

Operation

0xxx → Pc

ea → Pc

Assembler Syntax

JMP	xxx
JMP	ea

Instruction Fields

{xxx}	**aaaaaaaaaaaa**	Short Jump Address
{ea}	**MMMRRR**	Effective Address (see **Table 12-13** on page 12-22)

Description Jump to the location in program memory given by the instruction's effective address. All memory-alterable addressing modes can be used for the effective address. A Fast Short Jump addressing mode can also be used. The 12-bit data is zero-extended to form the effective address.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

JMP ea

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	1	M	M	M	R	R	R	1	0	0	0	0	0	0	0
Optional Effective Address Extension																							

JMP xxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	0	0	0	0	a	a	a	a	a	a	a	a	a	a	a	a

JScc Jump to Subroutine Conditionally JScc

Operation

If cc, then SP + 1 → SP; PC → SSH;SR → SSL;0xxx → PC
else PC + 1 → PC

If cc, then SP + 1 → SP; PC → SSH;SR → SSL;ea → PC
else PC + 1 → PC

Assembler Syntax

JScc xxx

JScc ea

Instruction Fields

{cc}	**CCCC**	Condition code (see **Table 12-18** on page 12-28)
{xxx}	**aaaaaaaaaaaa**	Short Jump Address
{ea}	**MMMRRR**	Effective Address (see **Table 12-13** on page 12-22)

Description Jump to the subroutine whose location in program memory is given by the instruction's effective address if the specified condition is true. If the specified condition is true, the address of the instruction immediately following the JScc instruction (PC) and the SR are pushed onto the system stack. Program execution then continues at the specified effective address in program memory. If the specified condition is false, the PC is incremented, and any extension word is ignored. However, the address register specified in the effective address field is always updated independently of the specified condition. All memory-alterable addressing modes can be used for the effective address. A fast short jump addressing mode can also be used. The 12-bit data is zero-extended to form the effective address. The conditions specified by "cc" are listed on **Table 12-18** on page 12-28.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

		23 16	15 8	7 0
JScc	xxx	0 0 0 0 1 1 1 1	C C C C a a a a	a a a a a a a a
JScc	ea	0 0 0 0 1 0 1 1	1 1 M M M R R R	1 0 1 0 C C C C
				Optional Effective Address Extension

JSCLR Jump to Subroutine if Bit Clear JSCLR

Operation

If S{n} = 0	then	SP + 1 → SP;PC → SSH;SR → SSL; ;xxxx → PC	
	else	PC + 1 → PC	
If S{n} = 0	then	SP + 1 → SP;PC → SSH;SR → SSL; ;xxxx → PC	
	else	PC + 1 → PC	
If S{n} = 0	then	SP + 1 → SP;PC → SSH;SR → SSL; ;xxxx → PC	
	else	PC + 1 → PC	
If S{n} = 0	then	SP + 1 → SP;PC → SSH;SR → SSL; ;xxxx → PC	
	else	PC + 1 → PC	
If S{n} = 0	then	SP + 1 → SP;PC → SSH;SR → SSL; ;xxxx fiPC	
	else	PC + 1 → PC	

Assembler Syntax

JSCLR	#n,[X or Y]:ea,xxxx
JSCLR	#n,[X or Y],aa,xxxx
JSCLR	#n,[X or Y]:pp,xxxx
JSCLR	#n,[X or Y]:qq,xxxx
JSCLR	#n,S,xxxx

Instruction Fields

{#n}	**bbbb**	Bit number [0–23]	
{ea}	**MMMRRR**	Effective Address	
{X/Y}	**S**	Memory Space [X,Y]	
{xxxx}		24-bit absolute Address extension word	
{aa}	**aaaaaa**	Absolute Address [0–63]	See **Table 12-13** on page 12-22
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0–$FFFFFF]	
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80–$FFFFBF]	
{S}	**DDDDDD**	Source register [all on-chip registers]	

Description Jump to the subroutine at the 24-bit absolute address in program memory specified in the instruction's 24-bit extension word if the n^{th} bit of the source operand S is clear. The bit to be tested is selected by an immediate bit number from 0–23. If the n^{th} bit of source operand S is clear, the address of the instruction immediately following the JSCLR instruction (PC) and the SR are pushed onto the system stack. Program execution then continues at the specified absolute address in the instruction's 24-bit extension word. If the specified memory bit is not clear, the PC is incremented and the extension word is ignored. However, the address register specified in the effective address field is always updated independently of the state of the n^{th} bit. All address register indirect addressing modes can reference the source operand S. Absolute short and I/O short addressing modes can also be used.

Condition Codes

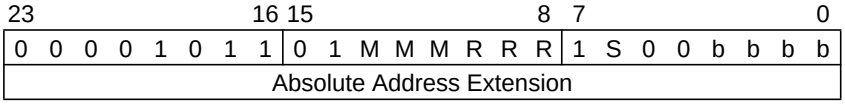
7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.

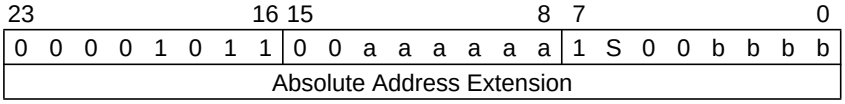
— Unchanged by the instruction.

Instruction Formats and opcodes

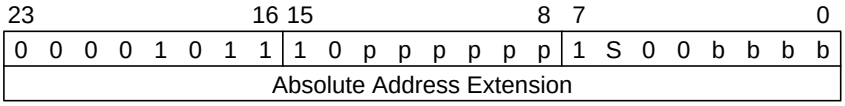
JSCLR #n,[X or Y]:ea,xxxx



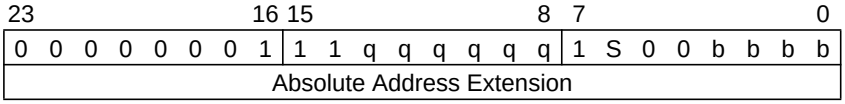
JSCLR #n,[X or Y]:aa,xxxx



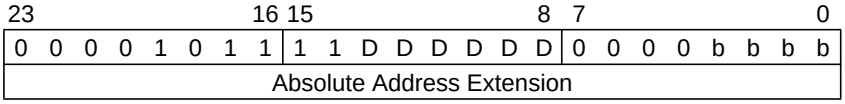
JSCLR #n,[X or Y]:pp,xxxx



JSCLR #n,[X or Y]:qq,xxxx



JSCLR #n,S,xxxx



JSET — Jump if Bit Set — JSET

Operation

If	S{n} = 1	then xxxx → PC else PC + 1 → PC
If	S{n} = 1	then xxxx → PC else PC + 1 → PC
If	S{n} = 1	then xxxx → PC else PC + 1 → PC
If	S{n} = 1	then xxxx → PC else PC + 1 → PC
If	S{n} = 1	then xxxx → PC else PC + 1 → PC

Assembler Syntax

JSET	#n,[X or Y]:ea,xxxx
JSET	#n,[X or Y],aa,xxxx
JSET	#n,[X or Y]:pp,xxxx
JSET	#n,[X or Y]:qq,xxxx
JSET	#n,S,xxxx

Instruction Fields

{#n}	bbbb	Bit number [0–23]
{ea}	MMMRRR	Effective Address (see **Table 12-13** on page 12-22)
{X/Y}	S	Memory Space [X,Y] (see **Table 12-13** on page 12-22)
{xxxx}		24-bit Absolute Address in extension word
{aa}	aaaaaa	Absolute Address [0 – 63]
{pp}	pppppp	I/O Short Address [64 addresses: $FFFFC0 – $FFFFFF]
{qq}	qqqqqq	I/O Short Address [64 addresses: $FFFF80 – $FFFFBF]
{S}	DDDDDD	Source register [all on-chip registers] (see **Table 12-13** on page 12-22)

Description Jump to the 24-bit absolute address in program memory specified in the instruction's 24-bit extension word if the n^{th} bit of the source operand S is set. The bit to be tested is selected by an immediate bit number from 0–23. If the specified memory bit is not set, the Program Counter (PC) is incremented, and the absolute address in the extension word is ignored. However, the address register specified in the effective address field is always updated independently of the state of the n^{th} bit. All address register indirect addressing modes can be used to reference the source operand S. Absolute short and I/O short addressing modes can also be used.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and opcodes

JSET #n,[X or Y]:ea,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	1	M	M	M	R	R	R	1	S	1	0	b	b	b	b
Absolute Address Extension																							

JSET #n,[X or Y]:aa,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	a	a	a	a	a	a	1	S	1	0	b	b	b	b
Absolute Address Extension																							

JSET #n,[X or Y]:pp,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	0	p	p	p	p	p	p	1	S	1	0	b	b	b	b
Absolute Address Extension																							

JSET #n,[X or Y]:qq,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	0	q	q	q	q	q	q	1	S	1	0	b	b	b	b
Absolute Address Extension																							

JSET #n,S,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	1	D	D	D	D	D	D	0	0	1	0	b	b	b	b
Absolute Address Extension																							

JSR — Jump to Subroutine — JSR

Operation

SP + 1 → SP; PC → SSH; SR → SSL; 0xxx → PC

SP + 1 → SP; PC → SSH; SR → SSL; ea → PC

Assembler Syntax

JSR xxx

JSR ea

Instruction Fields

{xxx} **aaaaaaaaaaaa** Short Jump Address

{ea} **MMMRRR** Effective Address (see **Table 12-13** on page 12-22)

Description Jump to the subroutine whose location in program memory is given by the instruction's effective address. The address of the instruction immediately following the JSR instruction (PC) and the system Status Register (SR) is pushed onto the system stack. Program execution then continues at the specified effective address in program memory. All memory-alterable addressing modes can be used for the effective address. A fast short jump addressing mode can also be used. The 12-bit data is zero-extended to form the effective address.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

JSR ea

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	1	1	M	M	M	R	R	R	1	0	0	0	0	0	0	0
Optional Effective Address Extension																							

JSR xxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	0	0	0	0	a	a	a	a	a	a	a	a	a	a	a	a

JSSET Jump to Subroutine if Bit Set JSSET

Operation

		Assembler Syntax	
If S{n} = 1	then SP + 1 → SP;PC → SSH;SR → SSL; ;xxxx → PC else PC + 1 → PC	JSSET	#n,[X or Y]:ea,xxxx
If S{n}=1	then SP + 1 → SP;PC → SSH;SR → SSL; ;xxxx → PC else PC + 1 → PC	JSSET	#n,[X or Y],aa,xxxx
If S{n}=1	then SP + 1 → SP;PC → SSH;SR → SSL; ;xxxx → PC else PC + 1 → PC	JSSET	#n,[X or Y]:pp,xxxx
If S{n}=1	then SP + 1 → SP;PC → SSH;SR → SSL; ;xxxx → PC else PC + 1 → PC	JSSET	#n,[X or Y]:qq,xxxx
If S{n}=1	then SP + 1 → SP;PC → SSH;SR → SSL; ;xxxx → PC else PC + 1 → PC	JSSET	#n,S,xxxx

Instruction Fields

{#n}	**bbbb**	Bit number [0–23]
{ea}	**MMMRRR**	Effective Address (see **Table 12-13** on page 12-22)
{X/Y}	**S**	Memory Space [X,Y] (see **Table 12-13** on page 12-22)
{xxxx}		24-bit PC absolute Address extension word
{aa}	**aaaaaa**	Absolute Address [0–63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0–$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80–$FFFFBF]
{S}	**DDDDDD**	Source register [all on-chip registers] (see **Table 12-13** on page 12-22)

Description Jump to the subroutine at the 24-bit absolute address in program memory specified in the instruction's 24-bit extension word if the n^{th} bit of the source operand S is set. The bit to be tested is selected by an immediate bit number from 0–23. If the n^{th} bit of the source operand S is set, the address of the instruction immediately following the JSSET instruction (PC) and the system Status Register (SR) are pushed onto the system stack. Program execution then continues at the specified absolute address in the instruction's 24-bit extension word. If the specified memory bit is not set, the Program Counter (PC) is incremented, and the extension word is ignored. However, the address register specified in the effective address field is always updated independently of the

state of the n^{th} bit. All address register indirect addressing modes can be used to reference the source operand S. Absolute short and I/O short addressing modes can also be used.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and opcodes

JSSET #n,[X or Y]:ea,xxxx

23								16	15							8	7							0
0	0	0	0	1	0	1	1		0	1	M	M	M	R	R	R	1	S	1	0	b	b	b	b
Absolute Address Extension																								

JSSET #n,[X or Y]:aa,xxxx

23								16	15							8	7							0
0	0	0	0	1	0	1	1		0	0	a	a	a	a	a	a	1	S	1	0	b	b	b	b
Absolute Address Extension																								

JSSET #n,[X or Y]:pp,xxxx

23								16	15							8	7							0
0	0	0	0	1	0	1	1		1	0	p	p	p	p	p	p	1	S	1	0	b	b	b	b
Absolute Address Extension																								

JSSET #n,[X or Y]:qq,xxxx

23								16	15							8	7							0
0	0	0	0	0	0	0	1		1	1	q	q	q	q	q	q	1	S	1	0	b	b	b	b
Absolute Address Extension																								

JSSET #n,S,xxxx

23								16	15							8	7							0
0	0	0	0	1	0	1	1		1	1	D	D	D	D	D	D	0	0	1	0	b	b	b	b
Absolute Address Extension																								

LRA — Load PC-Relative Address — LRA

Operation

PC + Rn → D

PC + xxxx → D

Assembler Syntax

LRA Rn,D

LRA xxxx,D

Instruction Fields

{Rn}	**RRR**	Address register [R0–R7]
{D}	**ddddd**	Destination address register [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B,R0–R7,N0–N7] (see **Table 12-16** on page 12-24)
{xxxx}		24-bit PC Long Displacement

Description The PC is added to the specified displacement and the result is stored in destination D. The displacement is a two's-complement 24-bit integer that represents the relative distance from the current PC to the destination PC. Long Displacement and Address Register PC-Relative addressing modes can be used. Note that if D is SSH, the SP is pre-incremented by one.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcode

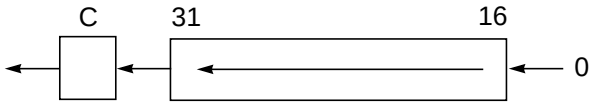
LRA Rn,D

23							16	15							8	7							0
0	0	0	0	0	1	0	0	1	1	0	0	0	R	R	R	0	0	0	d	d	d	d	d

LRA xxxx,D

23							16	15							8	7							0
0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	d	d	d	d	d
Long Displacement																							

Operation



Assembler Syntax

LSL D (parallel move)
LSL #ii,D
LSL S,D

Instruction Fields

{D}	**D**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{S}	**sss**	Control register [X0,X1,Y0,Y1,A1,B1] (see **Table 12-13** on page 12-22)
{#ii}	**iiiii**	5-bit unsigned integer [0–16] denoting the shift amount

Description

- Single-bit shift: Logically shift Bits 47–24 of the destination operand D one bit to the left and store the result in the destination accumulator. Prior to instruction execution, Bit 47 of D is shifted into the carry bit C, and a 0 is shifted into Bit 24 of the destination accumulator D.
- Multi-bit shift: The contents of bits 47–24 of the destination accumulator D are shifted left #ii bits. Bits shifted out of position 47 are lost, except for the last bit that is latched in the Carry bit. Zeros are supplied to the vacated positions on the right. The result is placed into bits 47–24 of the destination accumulator D. The number of bits to shift is determined by the 5-bit immediate field in the instruction, or by the unsigned integer located in the control register S. If a zero shift count is specified, the carry bit is cleared.

This is a 24-bit operation. The remaining bits of the destination accumulator are not affected. The number of shifts should not exceed the value of 24.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	*	*	*	*
CCR							

* N Set if Bit 47 of the result is set.

* Z Set if bits 47–24 of the result are 0.

* V Always cleared.

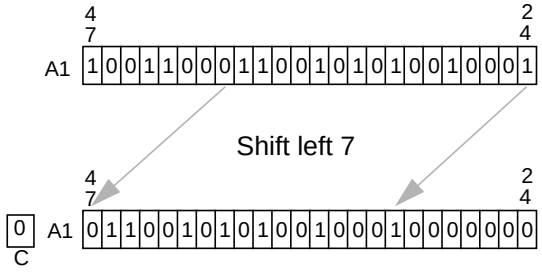
* C Set if the last bit shifted out of the operand is set, cleared for a shift count of 0, and cleared otherwise.

√ Changed according to the standard definition.

— Unchanged by the instruction.

Example

LSL #7, A



Instruction Formats and opcodes

		23 ... 8	7 ... 0
LSL	D	Data Bus Move Field	0 0 1 1 D 0 1 1
		Optional Effective Address Extension	

		23 ... 16	15 ... 8	7 ... 0
LSL	#ii,D	0 0 0 0 1 1 0 0	0 0 0 1 1 1 1 0	1 0 i i i i i D

		23 ... 16	15 ... 8	7 ... 0
LSL	S,D	0 0 0 0 1 1 0 0	0 0 0 1 1 1 1 0	0 0 0 1 s s s D

Operation

Assembler Syntax

LSR D (parallel move)
LSR #ii,D
LSR S,D

Instruction Fields

{D}	**D**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{S}	**sss**	Control register [X0,X1,Y0,Y1,A1,B1] (see **Table 12-13** on page 12-22)
{#ii}	**iiiii**	5-bit unsigned integer [0–23] denoting the shift amount

Description

- Single-bit shift: Logically shift bits 47–24 of the destination operand D one bit to the right and store the result in the destination accumulator. Prior to instruction execution, Bit 24 of D is shifted into the Carry bit (C), and a 0 is shifted into Bit 47 of the destination accumulator D.
- Multi-bit shift: The contents of bits 47–24 of the destination accumulator D are shifted right #ii bits. Bits shifted out of position 16 are lost except for the last bit that is latched in the C bit. Zeroes are supplied to the vacated positions on the left. The result is placed into bits 47–24 of the destination accumulator D. The number of bits to shift is determined by the 5-bit immediate field in the instruction, or by the unsigned integer located in the control register S. If a zero shift count is specified, the C bit is cleared.

This is a 24-bit operation. The remaining bits of the destination register are not affected. The number of shifts should not exceed the value of 24.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	*	*	*	*
CCR							

* N Set if Bit 47 of the result is set.

* Z Set if Bits 47–24 of the result are 0.

* V Always cleared.

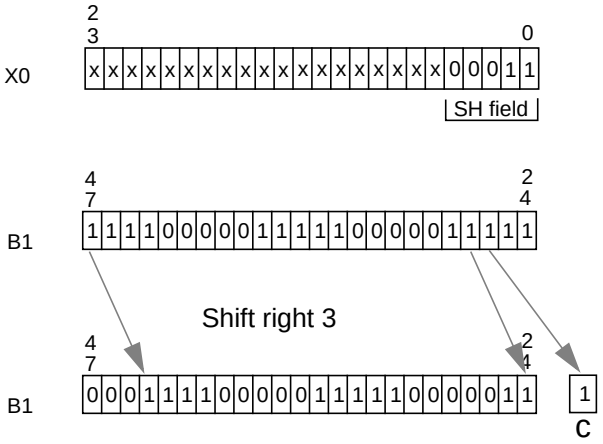
* C Set if the last bit shifted out of the operand is set, cleared for a shift count of zero, and cleared otherwise.

√ Changed according to the standard definition.

— Unchanged by the instruction.

Example

LSR X0,B



Instruction Formats and opcodes

		23 … 8	7 … 0
LSR	D	Data Bus Move Field	0 0 1 0 D 0 1 1
		Optional Effective Address Extension	

		23 … 16	15 … 8	7 … 0
LSR	#ii,D	0 0 0 0 1 1 0 0	0 0 0 1 1 1 1 0	1 1 i i i i i D

		23 … 16	15 … 8	7 … 0
LSR	S,D	0 0 0 0 1 1 0 0	0 0 0 1 1 1 1 0	0 0 1 1 s s s D

LUA Load Updated Address LUA

Operation

ea → D (No update performed)

Rn + aa → D

ea → D (No update performed)

Rn + aa → D

Assembler Syntax

LUA	ea,D
LUA	(Rn + aa),D
LEA	ea,D
LEA	(Rn + aa),D

Instruction Fields

{ea}	**MMRRR**	Effective address (see **Table 12-13** on page 12-22)
{D}	**ddddd**	Destination address register [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B,R0–R7,N0–N7] (see **Table 12-16** on page 12-24)
{D}	**dddd**	Destination address register [R0–R7,N0–N7] (see **Table 12-16** on page 12-24)
{aa}	**aaaaaaa**	7-bit sign extended short displacement address
{Rn}	**RRR**	Source address register [R0–R7]

Note: RRR refers to a source address register (R0–R7), while dddd/ddddd refer to a destination address register (R0–R7 or N0–N7).

Description Load the updated address into the destination address register D. The source address register and the update mode used to compute the updated address are specified by the effective address (ea). Only the following addressing modes can be used: Post + N, Post – N, Post + 1, Post – 1. Note that the source address register specified in the effective address is not updated. This is the only case where an address register is not updated, although stated otherwise in the effective address mode bits.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcode

LUA/LEA ea,D

23							16	15						8	7							0
0	0	0	0	0	1	0	0	0	1	0	M	M	R	R	R	0	0	0	d	d	d	d	d

LUA/LEA (Rn + aa),D

23							16	15						8	7							0
0	0	0	0	0	1	0	0	0	0	a	a	a	R	R	R	a	a	a	a	d	d	d	d

Note: LEA is a synonym for LUA. The simulator on-line disassembly translates the opcodes into LUA.

MAC — Signed Multiply Accumulate — MAC

Operation

D ±S1 * S2 → D (parallel move)

D ±S1 * S2 → D (parallel move)

D ±(S1 * 2^{-n}) → D (**no** parallel move)

Assembler Syntax

MAC (±)S1,S2,D (parallel move)

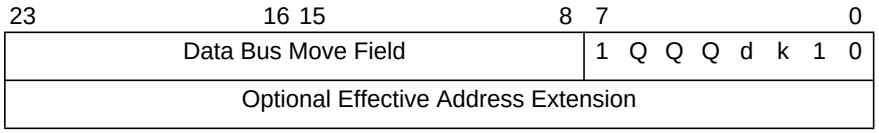
MAC (±)S2,S1,D (parallel move)

MAC (±)S,#n,D (**no** parallel move)

Instruction Formats and opcodes 1

MAC (±)S1,S2,D

MAC (±)S2,S1,D



Instruction Fields

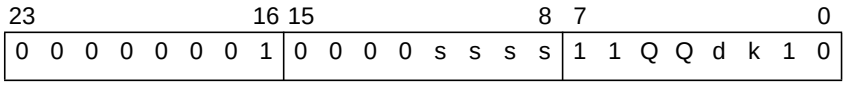
{S1,S2} **QQQ** Source registers S1,S2 [X0*X0,Y0*Y0,X1*X0,Y1*Y0,X0*Y1,Y0*X0,X1*Y0,Y1*X1] (see **Table 12-16** on page 12-24)

{D} **d** Destination accumulator [A,B] (see **Table 12-16** on page 12-24)

{±} **k** Sign [+,–] (see **Table 12-16** on page 12-24)

Instruction Formats and opcode 2

MAC (±)S,#n,D



Instruction Fields

{S} **QQ** Source register [Y1,X0,Y0,X1]] (see **Table 12-16** on page 12-24)

{D} **d** Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

{±} **k** Sign [+,–] (see **Table 12-16** on page 12-24)

{#n} **ssss** Immediate operand (see **Table 12-16** on page 12-24)

Description Multiply the two signed 24-bit source operands S1 and S2 (**or** the signed 24-bit source operand S by the positive 24-bit immediate operand 2^{-n}) and add/subtract the product to/from the specified 56-bit destination accumulator D. The "–" sign option is used to negate the specified product prior to accumulation. The default sign option is "+".

Note that when the processor is in the Double Precision Multiply mode, the following instructions do not execute in the normal way and should only be used as part of the double precision multiply algorithm:

MAC X1, Y0, AMAC X1, Y0, B

MAC X0, Y1, AMAC X0, Y1, B

MAC Y1, X1, AMAC Y1, X1, B

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	—
CCR							

√ Changed according to the standard definition.
— Unchanged by the instruction.

Signed Multiply Accumulate With Immediate Operand

Operation

D ±#xxxx*S → D

Assembler Syntax

MACI (±)#xxxx,S,D

Instruction Fields

{S}	**qq**	Source register [X0,Y0,X1,Y1] (see **Table 12-16** on page 12-24)
{D}	**d**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{±}	**k**	Sign [+,–] (see **Table 12-16** on page 12-24)
#xxxxxx		24-bit Immediate Long Data extension word

Description Multiply the two signed 24-bit source operands #xxxx and S and add/subtract the product to/from the specified 56-bit destination accumulator D. The "–" sign option is used to negate the specified product prior to accumulation. The default sign option is "+".

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	√	√	√	√	√	√	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and opcode

MACI (±)#xxxx,S,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	q	q	d	k	1	0
Immediate Data Extension																							

Mixed Multiply Accumulate

Operation

D ±S1 * S2 → D (S1 unsigned, S2 unsigned)

D ±S1 * S2 → D (S1 signed, S2 unsigned)

Assembler Syntax

MACuu	(±)S1,S2,D (no parallel move)
MACsu	(±)S2,S1,D (no parallel move)

Instruction Fields

{S1,S2}	**QQQQ**	Source registers S1,S2 [all combinations of X0,X1,Y0 and Y1] (see **Table 12-16** on page 12-24)
{D}	**d**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{±}	**k**	Sign [+,–] (see **Table 12-16** on page 12-24)
{s}		[ss,us] (see **Table 12-16** on page 12-24)

Description Multiply the two 24-bit source operands S1 and S2 and add/subtract the product to/from the specified 56-bit destination accumulator D. One or two of the source operands can be unsigned. The "–" sign option is used to negate the specified product prior to accumulation. The default sign option is "+".

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	√	√	√	√	√	√	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and opcodes

MACsu (±)S1,S2,D

MACuu (±)S1,S2,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	1	s	d	k	Q	Q	Q	Q

MACR Signed Multiply Accumulate and Round MACR

Operation

D ±S1 * S2 + r → D (parallel move)

D ±S1 * S2 + r → D (parallel move)

D ±(S1 * 2^{-n}) + r → D (**no** parallel move)

Assembler Syntax

MACR (±)S1,S2,D (parallel move)

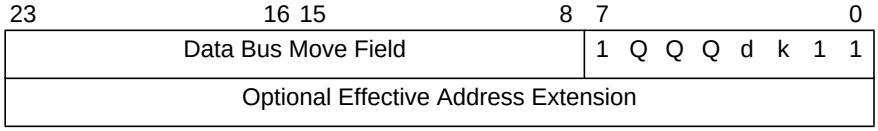
MACR (±)S2,S1,D (parallel move)

MACR (±)S,#n,D (**no** parallel move)

Instruction Formats and opcodes 1

MACR (±)S1,S2,D

MACR (±)S2,S1,D



Instruction Fields

{S1,S2} **QQQ** Source registers S1,S2 [X0*X0,Y0*Y0,X1*X0,Y1*Y0,X0*Y1,Y0*X0,X1*Y0,Y1*X1]

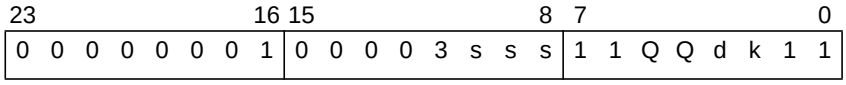
(see **Table 12-16** on page 12-24)

{D} **d** Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

{±} **k** Sign [+,–] (see **Table 12-16** on page 12-24)

Instruction Formats and opcode 2

MACR (±)S,#n,D



Instruction Fields

{S} **QQ** Source register [Y1,X0,Y0,X1] (see **Table 12-16** on page 12-24)

{D} **d** Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

{±} **k** Sign [+,–] (see **Table 12-16** on page 12-24)

{#n} **ssss** Immediate operand (see **Table 12-16** on page 12-24)

Description Multiply the two signed 24-bit source operands S1 and S2 (or the signed 24-bit source operand S by the positive 24-bit immediate operand 2^{-n}), add/subtract the product to/from the specified 56-bit destination accumulator D, and round the result using either convergent or two's-complement rounding. The rounded result is stored in destination accumulator D. The "–" sign option negates the specified product prior to accumulation. The default sign option is "+." The LSB of the result is rounded into the upper portion of the destination accumulator. Once rounding is complete, the LSBs of

destination accumulator D are loaded with 0s to maintain an unbiased accumulator value that the next instruction can reuse. The upper portion of the accumulator contains the rounded result that can be read out to the data buses. Refer to the RND instruction for details on the rounding process.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Signed MAC and Round With Immediate Operand

Operation

D ±#xxxxxx * S → D

Assembler Syntax

MACRI (±)#xxxxxx,S,D

Instruction Fields

{S}	**qq**	Source register [X0,Y0,X1,Y1] (see **Table 12-16** on page 12-24)
{D}	**d**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{±}	**k**	Sign [+,-] (see **Table 12-16** on page 12-24)
#xxxx		24-bit Immediate Long Data extension word

Description Multiply the two signed 24-bit source operands #xxxx and S, add/subtract the product to/from the specified 56-bit destination accumulator D, and then round the result using either convergent or two's-complement rounding. The rounded result is stored in the destination accumulator D. The "–" sign option negates the specified product prior to accumulation. The default sign option is "+". The contribution of the LSBs of the result is rounded into the upper portion of the destination accumulator. Once rounding is complete, the LSBs of the destination accumulator D are loaded with 0s to maintain an unbiased accumulator value that the next instruction can reuse. The upper portion of the accumulator contains the rounded result that can be read out to the data buses. Refer to the RND instruction for details on the rounding process.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	√	√	√	√	√	√	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and opcode

MACRI (±)#xxxx,S,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	q	q	d	k	1	1
Immediate Data Extension																							

Operation

If B – A $\leq$ 0 then A $\rightarrow$ B

Assembler Syntax

MAX A,B (parallel move)

Description Subtract the signed value of the source accumulator from the signed value of the destination accumulator. If the difference is negative or 0, (A $\geq$ B) then transfer the source accumulator to destination accumulator. Otherwise, do not change the destination accumulator. This is a 56-bit operation. Note that the Carry bit signifies a transfer has been performed.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	*
CCR							

* C This bit is cleared if the conditional transfer is performed, and set otherwise.
√ Changed according to the standard definition.
— Unchanged by the instruction.

Instruction Formats and opcodes

MAX A, B

23 16	15 8	7 0
Data Bus Move Field		0 0 0 1 1 1 0 1
Optional Effective Address Extension		

Operation

If |B| – |A| $\leq$ 0 then A $\rightarrow$ B

Assembler Syntax

MAXM A,B (parallel move)

Description Subtract the absolute value (magnitude) of the source accumulator from the absolute value of the destination accumulator. If the difference is negative or 0 (|A| $\geq$ |B|), then transfer the source accumulator to the destination accumulator. Otherwise, do not change the destination accumulator. This is a 56-bit operation. Note that the Carry bit (C) signifies a transfer has been performed.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	*
CCR							

* C This bit is cleared if the conditional transfer was performed, and set otherwise.

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and Opcodes

MAXM A, B

23 16	15 8	7 4	3 0
Data Bus Move Field		0 0 0 1	0 1 0 1
Optional Effective Address Extension			

MERGE Merge Two Half Words MERGE

Operation

{S[7:0],D[35:24]} → D[47:24]

Assembler Syntax

MERGE S,D

Instruction Fields

{D}	D	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{S}	SSS	Source register [X0,X1,Y0,Y1,A1,B1] (see **Table 12-16** on page 12-24)

Description The contents of bits 11–0 of the source register are concatenated to the contents of bits 35–24 of the destination accumulator. The result is stored in the destination accumulator. This instruction is a 24-bit operation. The remaining bits of the destination accumulator D are not affected.

Note:

1. MERGE can be used in conjunction with EXTRACT or INSERT instructions to concatenate width and offset fields into a control word.
2. In Sixteen-bit Arithmetic mode, the contents of bits 15-8 of the source register are concatenated with the contents of bits 39-32 of the destination accumulator. The result is placed in bits 47-32 of the destination accumulator.

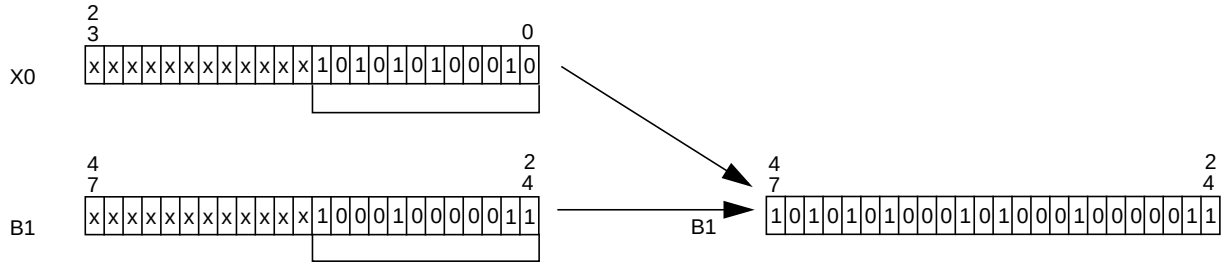
Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	*	*	*	—
CCR							

*	N	Set if bit 47 of the result is set.
*	Z	Set if bits 47–24 of the result are 0.
*	V	Always cleared.
—		Unchanged by the instruction.

Example

MERGE X0,B



Instruction Formats and Opcodes

		23								16	15							8	7							0
MERGE	S,D	0	0	0	0	1	1	0	0		0	0	0	1	1	0	1	1	1	0	0	0	S	S	S	D

The DSP56300 (family) core provides a set of MOVE instructions. **Table 12-14** lists these instructions, which are fully described in the following pages.

Table 12-14. Move Instructions

Instruction	Description	Page
MOVE	Move Data	**page 12-110**
	NO Parallel Data Move	**page 12-112**
I	Immediate Short Data Move	**page 12-113**
R	Register-to-Register Data Move	**page 12-116**
U	Address Register Update	**page 12-117**
X:	X Memory Data Move	**page 12-118**
X: R	X Memory and Register Data Move	**page 12-120**
Y	Y Memory Data Move	**page 12-122**
R: Y	Register and Y Memory Data Move	**page 12-124**
L:	Long Memory Data Move	**page 12-126**
X: Y	X Memory Data Move	**page 12-128**

Operation

S → D

Assembler Syntax

MOVE S,D

Description Move the contents of the specified data source S to the specified destination D. This instruction is equivalent to a Data ALU NOP with a parallel data move.

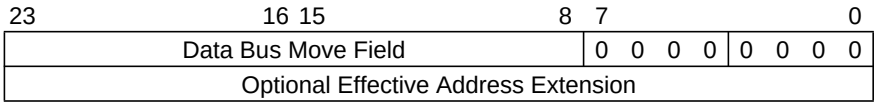
Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.
— Unchanged by the instruction.

Instruction Formats and Opcodes

MOVE S,D



Instruction Fields/

Parallel Move Description Thirty of the sixty-two instructions allow an optional parallel data bus movement over the X and/or Y data bus. This allows a Data ALU operation to be executed in parallel with up to two data bus moves during the instruction cycle. Ten types of parallel moves are permitted, including register-to-register moves, register-to-memory moves, and memory-to-register moves. However, not all addressing modes are allowed for each type of memory reference. The following section contains detailed descriptions about each type of parallel move operation.

NO Parallel Data Move

Operation

(. . .)

Assembler Syntax

(. . .)

where (. . .) refers to any arithmetic or logical instruction that allows parallel moves

Description Many instructions in the instruction set allow parallel moves. The parallel moves have been divided into ten opcode categories. This category is a parallel move NOP and does not involve data bus move activity.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and Opcodes

(. . .)

23							16	15							8	7	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	Instruction opcode	

Instruction Format (defined by instruction)

Immediate Short Data Move

Operation

(. . .), #xx → D

Assembler Syntax

(. . .) #xx,D

where (. . .) refers to any arithmetic or logical instruction that allows parallel moves

Instruction Fields

{#xx}	**iiiiiiii**	8-bit Immediate Short Data
{D}	**ddddd**	Destination register [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B,R0–R7,N0–N7] (see **Table 12-13** on page 12-22)

Description Move the 8-bit immediate data value (#xx) into the destination operand D. If the destination register D is A0, A1, A2, B0, B1, B2, R0–R7, or N0–N7, the 8-bit immediate short operand is interpreted as an *unsigned integer* and is stored in the specified destination register. That is, the 8-bit data is stored in the eight LSBs of the destination operand and the remaining bits of the destination operand D are zeroed. If the destination register D is X0, X1, Y0, Y1, A, or B, the 8-bit immediate short operand is interpreted as a *signed fraction* and is stored in the specified destination register. That is, the 8-bit data is stored in the eight MSBs of the destination operand and the remaining bits of the destination operand D are zeroed.

If the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator cannot be specified as a destination D in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction cannot specify A0, A1, A2, or A as its destination D. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction cannot specify B0, B1, B2, or B as its destination D. That is, duplicate destinations are *not* allowed within the same instruction.

Condition Codes

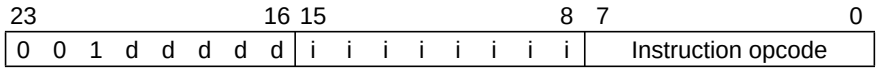
7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Immediate Short Data Move

Instruction Formats and Opcodes

(. . .) #xx,D



Operation

(. . .); S → D

Assembler Syntax

(. . .) S,D

where (. . .) refers to any arithmetic or logical instruction that allows parallel moves.

Instruction Fields

{S}	**eeeee**	Source register [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A, B,R0–R7,N0–N7]	See **Table 12-13** on page 12-22
{D}	**ddddd**	Destination register [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A, B,R0–R7,N0–N7]	

Description Move the source register S to the destination register D. If the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator cannot be specified as a destination D in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction cannot specify A0, A1, A2, or A as its destination D. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction cannot specify B0, B1, B2, or B as its destination D. That is, duplicate destinations are *not* allowed within the same instruction.

If the opcode-operand portion of the instruction specifies a given source or destination register, that same register or portion of that register can be used as a source S in the parallel data bus move operation. This allows data to be moved in the same instruction in which a Data ALU operation is using it as a source operand. That is, duplicate sources are allowed within the same instruction. Note that the MOVE A,B operation results in a 24-bit positive or negative saturation constant being stored in the B1 portion of the B accumulator if the signed integer portion of the A accumulator is in use.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and Opcodes

(. . .) S,D

23							16	15							8	7							0
0	0	1	0	0	0	e	e	e	e	e	e	d	d	d	d	d	Instruction opcode						

Operation

(. . .); ea⇒Rn

Assembler Syntax

(. . .) ea

where (. . .) refers to any arithmetic or logical instruction that allows parallel moves

Instruction Fields

{ea}	MMRRR	Effective Address (see **Table 12-13** on page 12-22)

Description Update the specified address register according to the specified effective addressing mode. All update addressing modes can be used.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and Opcodes

(. . .) ea

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	0	0	1	0	M	M	R	R	R	Instruction opcode							

X: X Memory Data Move X:

Operation

(. . .); X:ea → D	
(. . .); X:aa → D	
(. . .); S → X:ea	
(. . .); S → X:aa	
X:(Rn + xxx) → D	
X:(Rn + xxxx) → D	
D → X:(Rn + xxx)	
D → X:(Rn + xxxx)	

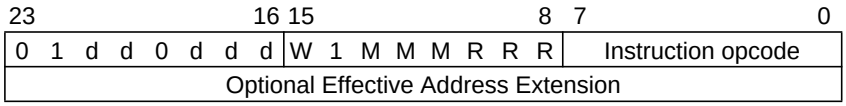
Assembler Syntax

(. . .)	X:ea,D
(. . .)	X:aa,D
(. . .)	S,X:ea
(. . .)	S,X:aa
MOVE	X:(Rn + xxx),D
MOVE	X:(Rn + xxxx),D
MOVE	D,X:(Rn + xxx)
MOVE	D,X:(Rn + xxxx)

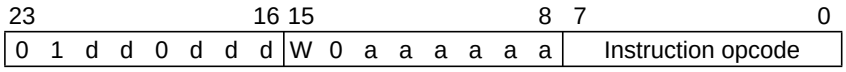
where (. . .) refers to any arithmetic or logical instruction that allows parallel moves.

Instruction Formats and Opcodes 1

(. . .) X:ea,D
(. . .) S,X:ea
(. . .) #xxxxxx,D



(. . .) X:aa,D
(. . .) S,X:aa



Instruction Fields

{ea} **MMMRRR** Effective Address (see **Table 12-13** on page 12-22)

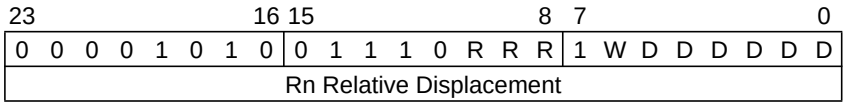
W Read S / Write D bit (see **Table 12-16** on page 12-24)

{S,D} **ddddd** Source/Destination registers [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B,R0–R7,N0–N7] (see **Table 12-13** on page 12-22)

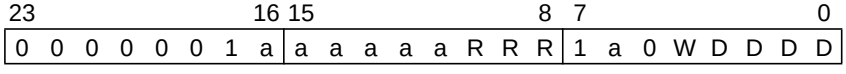
{aa} **aaaaaa** 6-bit Absolute Short Address

Instruction Formats and Opcodes 2

MOVE X:(Rn + xxxx),D
MOVE S,X:(Rn + xxxx)



MOVE X:(Rn + xxx),D
MOVE S,X:(Rn + xxx)



Instruction Fields

	W	Read S / Write D bit (see **Table 12-16** on page 12-24)
{xxx}	**aaaaaaa**	7-bit sign extended Short Displacement Address
{Rn}	**RRR**	Address register (R0–R7)
{D}	**DDDD**	Source/Destination registers [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B] (see **Table 12-16** on page 12-24)
{S,D}	**DDDDDD**	Source/Destination registers [all on-chip registers] (see **Table 12-13** on page 12-22)

Description Move the specified word operand from/to X memory. All memory addressing modes can be used, including absolute addressing and 24-bit immediate data. Absolute short addressing can also be used. If the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator cannot be specified as a destination D in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction cannot specify A0, A1, A2, or A as its destination D. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction cannot specify B0, B1, B2, or B as its destination D. That is, duplicate destinations are *not* allowed within the same instruction.

If the opcode-operand portion of the instruction specifies a given source or destination register, that same register or portion of that register can be used as a source S in the parallel data bus move operation. This allows data to be moved in the same instruction in which it is being used as a source operand by a Data ALU operation. That is, duplicate sources are allowed within the same instruction. As a result of the MOVE A,X:ea operation, a 24-bit positive or negative saturation constant is stored in the specified 24-bit X memory location if the signed integer portion of the A accumulator is in use.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Operation

Class I
(. . .); X:ea → D1; S2 → D2

(. . .); S1 → X:ea; S2 → D2

(. . .); #xxxxxx → D1; S2 → D2

Class II
(. . .); A → X:ea; X0 → A

(. . .); B → X:ea; X0 → B

Assembler Syntax

(. . .) X:ea,D1 S2,D2

(. . .) S1,X:ea S2,D2

(. . .) #xxxxxx,D1 S2,D2

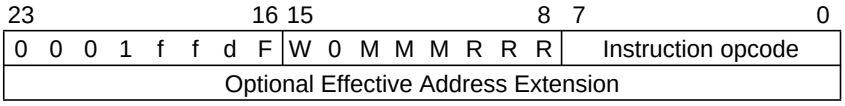
(. . .) A,X:ea X0,A

(. . .) B,X:ea X0,B

where (. . .) refers to any arithmetic or logical instruction that allows parallel moves

Class I Instruction Formats and Opcodes

(. . .) X:ea,D1 S2,D2
(. . .) S1,X:ea S2, D2
(. . .) #xxxx,D1 S2,D2

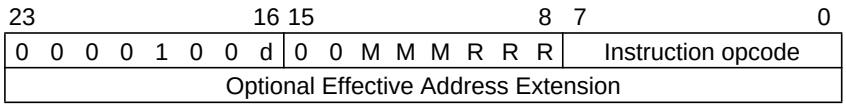


Instruction Fields

{ea}	**MMMRRR**	Effective Address (see **Table 12-13** on page 12-22)
	W	Read S1/Write D1 bit (see **Table 12-16** on page 12-24)
{S1,D1}	**ff**	S1/D1 register [X0,X1,A,B] (see **Table 12-16** on page 12-24)
{S2}	**d**	S2 accumulator [A,B] (see **Table 12-13** on page 12-22)
{D2}	**F**	D2 input register [Y0,Y1] (see **Table 12-16** on page 12-24)

Class II Instruction Formats and Opcodes

(. . .) A → X:ea X0 → A
(. . .) B → X:ea X0 → B



Instruction Fields

{ea}	**MMMRRR**	Effective Address (see **Table 12-13** on page 12-22)
	d	Move opcode (see **Table 12-16** on page 12-24)

Description

- Class I: Move a one-word operand from/to X memory and move another word operand from an accumulator (S2) to an input register (D2). All memory addressing modes, including absolute addressing and 24-bit immediate data, can be used. The register-to-register move (S2,D2) allows a Data ALU accumulator to be moved to a Data ALU input register for use as a Data ALU operand in the following instruction.
- Class II: Move one-word operand from a Data ALU accumulator to X memory and one-word operand from Data ALU register X0 to a Data ALU accumulator. One effective address is specified. All memory addressing modes except long absolute addressing and long immediate data can be used.

For both Class I and Class II X:R parallel data moves, if the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator cannot be specified as a destination D1 in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 40-bit A accumulator as its destination, the parallel data bus move portion of the instruction cannot specify A0, A1, A2, or A as its destination D1. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction cannot specify B0, B1, B2, or B as its destination D1. That is, duplicate destinations are *not* allowed within the same instruction. If the opcode-operand portion of the instruction specifies a given source or destination register, that same register or portion of that register can be used as a source S1 and/or S2 in the parallel data bus move operation. This allows data to be moved in the same instruction in which a Data ALU operation is using it as a source operand. That is, duplicate sources are allowed within the same instruction—S1 and S2 can specify the same register.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Operation	Assembler Syntax	
(. . .); Y:ea → D	(. . .)	Y:ea,D
(. . .); Y:aa → D	(. . .)	Y:aa,D
(. . .); S → Y:ea	(. . .)	S,Y:ea
(. . .); S → Y:aa	(. . .)	S,Y:aa
Y:(Rn + xxx) → D	MOVE	Y:(Rn + xxx),D
Y:(Rn + xxxx) → D	MOVE	Y:(Rn + xxxx),D
D → Y:(Rn + xxx)	MOVE	D,Y:(Rn + xxx)
D → Y:(Rn + xxxx)	MOVE	D,Y:(Rn + xxxx)

where (. . .) refers to any arithmetic or logical instruction that allows parallel moves

Instruction Formats and Opcodes 1

(. . .) Y:ea,D
(. . .) S,Y:ea
(. . .) #xxxx,D

23								16	15							8	7							0
0	1	d	d	1	d	d	d		W	1	M	M	M	R	R	R	Instruction opcode							
Optional Effective Address Extension																								

(. . .) Y:aa,D
(. . .) S,Y:aa

23								16	15							8	7							0
0	1	d	d	1	d	d	d		W	0	a	a	a	a	a	a	Instruction opcode							

Instruction Fields

{ea} **MMMRRR** Effective Address (see **Table 12-13** on page 12-22)

W Read S/Write D bit (see **Table 12-16** on page 12-24)

{S,D} **ddddd** Source/Destination registers [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B,R0–R7,N0–N7] (see **Table 12-13** on page 12-22)

{aa} **aaaaaa** Absolute Short Address

Instruction Formats and Opcodes 2

MOVE Y:(Rn + xxxx),D
MOVE D,Y:(Rn + xxxx)

23								16	15							8	7							0
0	0	0	0	1	0	1	1		0	1	1	1	0	R	R	R	1	W	D	D	D	D	D	D
Rn Relative Displacement																								

MOVE Y:(Rn + xxx),D
MOVE D,Y:(Rn + xxx)

23								16	15							8	7							0
0	0	0	0	0	0	1	a		a	a	a	a	a	R	R	R	1	a	1	W	D	D	D	D

Instruction Fields

	W	Read S/Write D bit (see **Table 12-16** on page 12-24)
{xxx}	**aaaaaaa**	7-bit sign extended Short Displacement Address
{Rn}	**RRR**	Address register (R0–R7)
{D}	**DDDD**	Source/Destination registers [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B] (see **Table 12-16** on page 12-24)
{S,D}	**DDDDDD**	Source/Destination registers [all on-chip registers] (see **Table 12-13** on page 12-22)

Description Move the specified word operand from/to Y memory. All memory addressing modes can be used, including absolute addressing, absolute short addressing, and 24-bit immediate data. If the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator cannot be specified as a destination D in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction cannot specify A0, A1, A2, or A as its destination D. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction cannot specify B0, B1, B2, or B as its destination D. That is, duplicate destinations are *not* allowed within the same instruction. If the opcode-operand portion of the instruction specifies a given source or destination register, that same register or portion of that register can be used as a source S in the parallel data bus move operation. This allows data to be moved in the same instruction in which a Data ALU operation is using it as a source operand. That is, duplicate sources are allowed within the same instruction. As a result of the MOVE A,Y:ea operation, a 24-bit positive or negative saturation constant is stored in the specified 24-bit Y memory location if the signed integer portion of the A accumulator is in use.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.

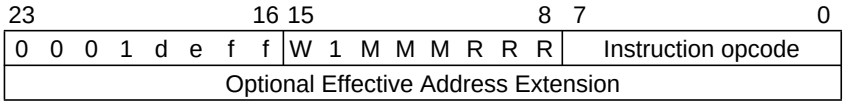
— Unchanged by the instruction.

Operation	Assembler Syntax	
Class I		
(. . .); S1 → D1; Y:ea → D2	(. . .)	S1,D1 Y:ea,D2
(. . .); S1 → D1; S2 → Y:ea	(. . .)	S1,D1 S2,Y:ea
(. . .); S1 → D1; #xxxxxx → D2	(. . .)	S1,D1 #xxxxxx,D2
Class II		
(. . .); Y0 → A; A → Y:ea	(. . .)	Y0,A A,Y:ea
(. . .); Y0 → B; B → Y:ea	(. . .)	Y0,B B,Y:ea

where (. . .) refers to any arithmetic or logical instruction that allows parallel moves

Class I Instruction Formats and Opcodes

(. . .) S1,D1 Y:ea,D2
(. . .) S1,D1 S2,Y:ea
(. . .) S1,D1 #xxxx,D2

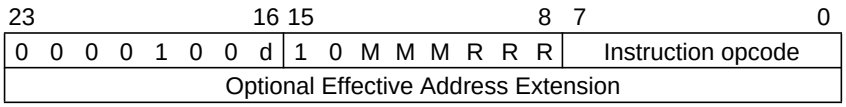


Instruction Fields

{ea}	**MMMRRR**	Effective Address	See **Table 12-13** on page 12-22
	W	Read S2/Write D2 bit	
{S1}	**d**	S1 accumulator [A,B]	**Table 12-16** on page 12-24
{D1}	**e**	D1 input register [X0,X1]	
{S2,D2}	**ff**	S2/D2 register [Y0,Y1,A,B]	

Class II Instruction Formats and opcodes

(. . .) Y0 → A A → Y:ea
(. . .) Y0 → B B → Y:ea



Instruction Fields

MMMRRR	ea = 6-bit Effective Address (see **Table 12-13** on page 12-22)
d	Move opcode (see **Table 12-16** on page 12-24)

Description

- Class I: Move a one-word operand from an accumulator (S1) to an input register (D1) and move another word operand from/to Y memory. All memory addressing modes, including absolute addressing and 16-bit immediate data, can be used. The register to register move (S1,D1) allows a Data ALU accumulator to be moved to a Data ALU input register for use as a Data ALU operand in the following instruction.
- Class II: Move a one-word operand from a Data ALU accumulator to Y memory and a one-word operand from Data ALU register Y0 to a Data ALU accumulator. One effective address is specified. All memory addressing modes, excluding long absolute addressing and long immediate data, can be used.

For both Class I and Class II R:Y parallel data moves, if the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator cannot be specified as a destination D2 in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction cannot specify A0, A1, A2, or A as its destination D2. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction cannot specify B0, B1, B2, or B as its destination D2. That is, duplicate destinations are *not* allowed within the same instruction. If the opcode-operand portion of the instruction specifies a given source or destination register, that same register or portion of that register can be used as a source S1 and/or S2 in the parallel data bus move operation. This allows data to be moved in the same instruction in which it is being used as a source operand by a Data ALU operation. That is, duplicate sources are allowed within the same instruction. Note that S1 and S2 can specify the same register.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

L: Long Memory Data Move L:

Operation

(. . .); X:ea → D1; Y:ea → D2

(. . .); X:aa → D1; Y:aa → D2

(. . .); S1 → X:ea; S2 → Y:ea

(. . .); S1 → X:aa; S2 → Y:aa

Assembler Syntax

(. . .) L:ea,D

(. . .) L:aa,D

(. . .) S,L:ea

(. . .) S,L:aa

where (. . .) refers to any arithmetic or logical instruction that allows parallel moves

Instruction Fields

{ea}	**MMMRRR**	Effective Address	**Table 12-13** on page 12-22
	W	Read S/Write D bit	
{L}	**LLL**	Two Data ALU registers	See **Table 12-16** on page 12-24
{aa}	**aaaaaa**	Absolute Short Address	

Description Move one 48-bit long-word operand from/to X and Y memory. Two Data ALU registers are concatenated to form the 48-bit long-word operand. This allows efficient moving of both double-precision (high:low) and complex (real:imaginary) data from/to one effective address in L (X:Y) memory. The same effective address is used for both the X and Y memory spaces; thus, only one effective address is required. Note that the A, B, A10, and B10 operands reference a single 48-bit signed (double-precision) quantity while the X, Y, AB, and BA operands reference two separate (i.e., real and imaginary) 24-bit signed quantities. All memory alterable addressing modes can be used. Absolute short addressing can also be used.

If the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator cannot be specified as a destination D in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction cannot specify A, A10, AB, or BA as destination D. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction cannot specify B, B10, AB, or BA as its destination D. That is, duplicate destinations are *not* allowed within the same instruction. If the opcode-operand portion of the instruction specifies a given source or destination register, that same register or portion of that register can be used as a source S in the parallel data bus move operation. This allows data to be moved in the same instruction in which it is being used as a source operand by a Data ALU operation. That is, duplicate sources are allowed within the same

instruction. Note that the operands A10, B10, X, Y, AB, and BA can be used only for a 32-bit long memory move as previously described. These operands cannot be used in any other type of instruction or parallel move.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.
— Unchanged by the instruction.

As a result of the MOVE A,L:ea operation, a 48-bit positive or negative saturation constant is stored in the specified 24-bit X and Y memory locations if the signed integer portion of the A accumulator is in use. As a result of the MOVE AB,L:ea operation, either one or two 24-bit positive and/or negative saturation constant(s) are stored in the specified 24-bit X and/or Y memory location(s) if the signed integer portion of the A and/or B accumulator(s) is in use.

Instruction Formats and Opcodes

	23 16	15 8	7 0
(. . .) L:ea,D	0 1 0 0 L 0 L L	W 1 M M M R R R	Instruction opcode
(. . .) S,L:ea	Optional Effective Address Extension		

	23 16	15 8	7 0
(. . .) L:aa,D	0 1 0 0 L 0 L L	W 0 a a a a a a	Instruction opcode
(. . .) S,L:aa			

X: Y: XY Memory Data Move X: Y:

Operation	Assembler Syntax
(. . .); X:<eax> → D1; Y:<eay> → D2	(. . .) X:<eax>,D1 Y:<eay>,D2
(. . .); X:<eax> → D1; S2 → Y:<eay>	(. . .) X:<eax>,D1 S2,Y:<eay>
(. . .); S1 → X:<eax>; Y:<eay> → D2	(. . .) S1,X:<eax> Y:<eay>,D2
(. . .); S1 → X:<eax>; S2 → Y:<eay>	(. . .) S1,X:<eax> S2,Y:<eay>

where (. . .) refers to any arithmetic or logical instruction that allows parallel moves

Instruction Fields

{<eax>}	**MMRRR**	5-bit X Effective Address (R0–R3 or R4–R7)
{<eay>}	**mmrr**	4-bit Y Effective Address (R4–R7 or R0–R3)
{S1,D1}	**ee**	S1/D1 register [X0,X1,A,B]
{S2,D2}	**ff**	S2/D2 register [Y0,Y1,A,B]

MMRRR,mmrr,ee,ff		See **Table 12-13** on page 12-22
	W	X move Operation Control (See **Table 12-16** on page 12-24)
	w	Y move Operation Control (See **Table 12-16** on page 12-24)

Description Move a one-word operand from/to X memory and move another word operand from/to Y memory. Note that two independent effective addresses are specified (<eax> and <eay>) where one of the effective addresses uses the lower bank of address registers (R0–R3) while the other effective address uses the upper bank of address registers (R4–R7). All parallel addressing modes can be used.

If the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator cannot be specified as a destination D1 or D2 in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction cannot specify A as its destination D1 or D2. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction cannot specify B as its destination D1 or D2. That is, duplicate destinations are *not* allowed within the same instruction. D1 and D2 cannot specify the same register.

If the instruction specifies an access to an internal X I/O and internal Y I/O modules (reflected by the address of the X memory and the Y memory), only the access to the internal X I/O module is executed. The access to the Y I/O module is discarded.

If the opcode-operand portion of the instruction specifies a given source or destination register, that same register or portion of that register can be used as a source S1 and/or S2 in the parallel data bus move operation. This allows data to be moved in the same instruction in which it is being used as a source operand by a Data ALU operation. That is, duplicate sources are allowed within the same instruction. Note that S1 and S2 can specify the same register.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and Opcodes

(. . .) X:<eax>,D1 Y:<eay>,D2
(. . .) X:<eax>,D1 S2,Y:<eay>
(. . .) S1,X:<eax> Y:<eay>,D2
(. . .) S1,X:<eax> S2,Y:<eay>

23								16	15							8	7							0
1	w	m	m	e	e	f	f		W	r	r	M	M	R	R	R	Instruction opcode							

MOVEC Move Control Register MOVEC

Operation

[X or Y]:ea → D1	
[X or Y]:aa → D1	
S1 → [X or Y]:ea	
S1 → [X or Y]:aa	
S1 → D2	
S2 → D1	
#xxxx → D1	
#xx → D1	

Assembler Syntax

MOVE(C)	[Xor Y]:ea,D1
MOVE(C)	[Xor Y]:aa,D1
MOVE(C)	S1,[X or Y]:ea
MOVE(C)	S1,[X or Y]:aa
MOVE(C)	S1,D2
MOVE(C)	S2,D1
MOVE(C)	#xxxx,D1
MOVE(C)	#xx,D1

Instruction Fields

{ea}	**MMMRRR**	Effective Address	See **Table 12-13** on page 12-22
	W	Read S/Write D bit	
{X/Y}	**S**	Memory Space [X,Y]	
{S1,D1}	**ddddd**	Program Controller register [M0–M7, VBA, SR, OMR, SP, SSH,SSL,LA,LC]	See **Table 12-16** on page 12-24
{aa}	**aaaaaa**	aa = 6-bit Absolute Short Address	
{S2,D2}	**eeeeee**	S2/D2 register [all on-chip registers]	
{#xx}	**iiiiiiii**	#xx = 8-bit Immediate Short Data	

Description Move the contents of the specified source control register S1 or S2 to the specified destination, or move the specified source to the specified destination control register D1 or D2. The control registers S1 and D1 are a subset of the S2 and D2 register set and consist of the Address ALU modifier registers and the program controller registers. These registers can be moved to or from any other register or memory space. All memory addressing modes, as well as an Immediate Short Addressing mode, can be used.

If the System Stack register SSH is specified as a source operand, the Stack Pointer (SP) is post-decremented by 1 after SSH has been read. If SSH is specified as a destination operand, the SP is preincremented by 1 before SSH is written. This allows the system stack to be efficiently extended using software stack pointer operations.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
*	*	*	*	*	*	*	*
CCR							

For D1 or D2 = SR operand:

- * S Set according to bit 7 of the source operand.
- * L Set according to bit 6 of the source operand.
- * E Set according to bit 5 of the source operand.
- * U Set according to bit 4 of the source operand.
- * N Set according to bit 3 of the source operand.
- * Z Set according to bit 2 of the source operand.
- * V Set according to bit 1 of the source operand.
- * C Set according to bit 0 of the source operand.

For D1 and D2 ≠ SR operand:

- * S Set if data growth has been detected.
- * L Set if data limiting has occurred during the move.

Instruction Formats and Opcodes

		23 16	15 8	7 0
MOVE(C)	[X or Y]:ea,D1	0 0 0 0 0 1 0 1	W 1 M M M R R R	O S 1 d d d d d
MOVE(C)	S1,[X or Y]:ea			
MOVE(C)	#xxxx,D1	Optional Effective Address Extension		

		23 16	15 8	7 0
MOVE(C)	[X or Y]:aa,D1	0 0 0 0 0 1 0 1	W 0 a a a a a a	0 S 1 d d d d d
MOVE(C)	S1,[X or Y]:aa			

		23 16	15 8	7 0
MOVE(C)	S1,D2	0 0 0 0 0 1 0 0	W 1 e e e e e e	1 0 1 d d d d d
MOVE(C)	S2,D1			

		23 16	15 8	7 0
MOVE(C)	#xx,D1	0 0 0 0 0 1 0 1	i i i i i i i i	1 0 1 d d d d d

MOVEM Move Program Memory MOVEM

Operation

S → P:ea	MOVE(M) S,P:ea
S → P:aa	MOVE(M) S,P:aa
P:ea → D	MOVE(M) P:ea,D
P:aa → D	MOVE(M) P:aa,D

Assembler Syntax

Instruction Fields

{ea}	**MMMRRR**	Effective Address (see **Table 12-13** on page 12-22)
	W	Read S/Write D bit (see **Table 12-16** on page 12-24)
{ S,D}	**dddddd**	Source/Destination register [all on-chip registers] (see **Table 12-13** on page 12-22)
{aa}	**aaaaaa**	Absolute Short Address

Description Move the specified operand from/to the specified Program (P) memory location. This is a powerful move instruction in that the source and destination registers S and D can be any register. All memory-alterable addressing modes can be used, as well as the Absolute Short Addressing mode. If the system stack register SSH is specified as a source operand, the system Stack Pointer (SP) is post-decremented by 1 after SSH has been read. If the system stack register SSH is specified as a destination operand, the SP is pre-incremented by 1 before SSH is written. This allows the system stack to be efficiently extended using software stack pointer operations.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
*	*	*	*	*	*	*	*
CCR							

For D1 or D2 = SR operand:

* S Set according to bit 7 of the source operand.
* L Set according to bit 6 of the source operand.
* E Set according to bit 5 of the source operand.
* U Set according to bit 4 of the source operand.
* N Set according to bit 3 of the source operand.
* Z Set according to bit 2 of the source operand.
* V Set according to bit 1 of the source operand.
* C Set according to bit 0 of the source operand.

For D1 and D2 ≠ SR operand:

* S Set if data growth has been detected.
* L Set if data limiting has occurred during the move.

Operation	Assembler Syntax	
S → P:ea	MOVE(M)	S,P:ea
S → P:aa	MOVE(M)	S,P:aa
P:ea → D	MOVE(M)	P:ea,D
P:aa → D	MOVE(M)	P:aa,D

Instruction Formats and Opcodes

		23 16	15 8	7 0
MOVE(M)	S,P:ea	0 0 0 0 0 1 1 1	W 1 M M M R R R	1 0 d d d d d d
MOVE(M)	P:ea,D	Optional Effective Address Extension		
MOVE(M)	S,P:aa	0 0 0 0 0 1 1 1	W 0 a a a a a a	0 0 d d d d d d
MOVE(M)	P:aa,D			

MOVEP Move Peripheral Data MOVEP

Operation

[X or Y]:pp → D

[X or Y]:qq → D

[X or Y]:pp → [X or Y]:ea

[X or Y]:qq → [X or Y]:ea

[X or Y]:pp → P:ea

[X or Y]:qq → P:ea

S → [X or Y]:pp

S → [X or Y]:qq

[X or Y]:ea → [X or Y]:pp

[X or Y]:ea → [X or Y]:qq

P:ea → [X or Y]:pp

P:ea → [X or Y]:qq

Assembler Syntax

MOVEP	[X or Y]:pp,D
MOVEP	[X or Y]:qq,D
MOVEP	[X or Y]:pp,[X or Y]:ea
MOVEP	[X or Y]:qq,[X or Y]:ea
MOVEP	[X or Y]:pp,P:ea
MOVEP	[X or Y]:qq,P:ea
MOVEP	S,[X or Y]:pp
MOVEP	S,[X or Y]:qq
MOVEP	[X or Y]:ea,[X or Y]:pp
MOVEP	[X or Y]:ea,[X or Y]:qq
MOVEP	P:ea,[X or Y]:pp
MOVEP	P:ea,[X or Y]:qq

Instruction Fields

{ea}	**MMMRRR**	Effective Address (see **Table 12-13** on page 12-22)
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0 – $FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80 – $FFFFBF]
{X/Y}	**S**	Memory space [X,Y] (see **Table 12-13** on page 12-22)
{X/Y}	**s**	Peripheral space [X,Y] (see **Table 12-13** on page 12-22)
	W	Read/write-peripheral (see **Table 12-13** on page 12-22)
{S,D}	**dddddd**	Source/Destination register [all on-chip registers] (see **Table 12-13** on page 12-22)

Description Move the specified operand to or from the specified X or Y I/O peripheral. The I/O Short Addressing mode is used for the I/O peripheral address. All memory addressing modes can be used for the X or Y memory effective address; all memory-alterable addressing modes can be used for the P memory effective address. All the I/O space ($FFFF80 – $FFFFFF) can be accessed, except for the P: reference opcode.If the System Stack register SSH is specified as a source operand, the system Stack Pointer (SP) is post-decremented by 1 after SSH has been read. If SSH is specified as a destination operand, the SP is pre-incremented by 1 before SSH is written. This allows the system stack to be efficiently extended using software stack pointer operations.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
*	*	*	*	*	*	*	*
CCR							

For D1 or D2 = SR operand:

- * S Set according to bit 7 of the source operand.
- * L Set according to bit 6 of the source operand.
- * E Set according to bit 5 of the source operand.
- * U Set according to bit 4 of the source operand.
- * N Set according to bit 3 of the source operand.
- * Z Set according to bit 2 of the source operand.
- * V Set according to bit 1 of the source operand.
- * C Set according to bit 0 of the source operand.

For D1 and D2 ≠ SR operand:

- * S Set if data growth is detected.
- * L Set if data limiting occurred during the move.

Instruction Formats and Opcodes

X: or Y: Reference (high I/O address)

MOVEP [X or Y]:pp,[X or Y]:ea
MOVEP [X or Y]:ea,[X or Y]:pp

23								16	15							8	7							0
0	0	0	0	1	0	0	s		W	1	M	M	M	R	R	R	1	S	p	p	p	p	p	p
Optional Effective Address Extension																								

X: or Y: Reference (low I/O address)

MOVEP X:qq,[X or Y]:ea
MOVEP [X or Y]:ea,X:qq

23								16	15							8	7							0
0	0	0	0	0	1	1	1		W	1	M	M	M	R	R	R	0	S	q	q	q	q	q	q
Optional Effective Address Extension																								

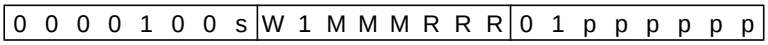
X: or Y: Reference (low I/O address)

MOVEP Y:qq,[X or Y]:ea
MOVEP [X or Y]:ea,Y:qq

23								16	15							8	7							0
0	0	0	0	0	1	1	1		W	0	M	M	M	R	R	R	1	S	q	q	q	q	q	q
Optional Effective Address Extension																								

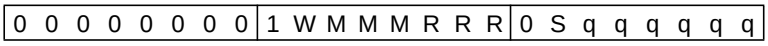
P: Reference (high I/O address)

MOVEP P:ea,[X or Y]:pp
MOVEP [X or Y]:pp,P:ea



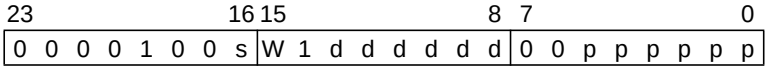
P: Reference (low I/O address)

MOVEP P:ea,[X or Y]:qq
MOVEP [X or Y]:qq,P:ea



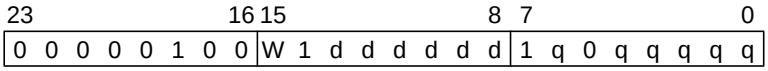
Register Reference (high I/O address)

MOVEP S,[X or Y]:pp
MOVEP [X or Y]:pp,D



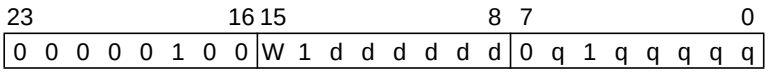
Register Reference: (low I/O address)

MOVEP S,X:qq
MOVEP X:qq,D



Register Reference: (low I/O address)

MOVEP S,Y:qq
MOVEP Y:qq,D



MPY — Signed Multiply — MPY

Operation		Assembler Syntax	
±S1 * S2 → D	(parallel move)	MPY (±)S1,S2,D	(parallel move)
±S1 * S2 → D	(parallel move)	MPY (±)S2,S1,D	(parallel move)
±(S1 * 2^{-n}) → D	(no parallel move)	MPY (±)S,#n,D	(no parallel move)

Instruction Fields 1

{S1,S2}	**QQQ**	Source registers S1,S2 [X0*X0, Y0*Y0, X1*X0, Y1*Y0, X0*Y1, Y0*X0, X1*Y0, Y1*X1] (see **Table 12-16** on page 12-24)
{D}	**d**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{±+/-}	**k**	Sign [+,–] (see **Table 12-16** on page 12-24)

Instruction Fields 2

{S}	**QQ**	Source register [Y1,X0,Y0,X1] (see **Table 12-16** on page 12-24)
{D}	**d**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{±}	**k**	Sign [+,–] (see **Table 12-16** on page 12-24)
{#n}	**sssss**	Immediate operand (see **Table 12-16** on page 12-24)

Description Multiply the two signed 24-bit source operands S1 and S2 and store the resulting product in the specified 56-bit destination accumulator D. Or, multiply the signed 24-bit source operand S by the positive 24-bit immediate operand 2^{-n} and store the resulting product in the specified 56-bit destination accumulator D. The “–” sign option is used to negate the specified product prior to accumulation. The default sign option is “+”. When the processor is in the Double-Precision Multiply mode, the following instructions do not execute in the normal way and should be used only as part of the double-precision multiply algorithm:

```
MPY Y0,X0,A     MPY Y0, X0,B
```

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	—
CCR							

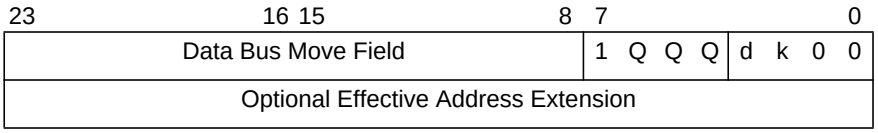
√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and Opcodes 1

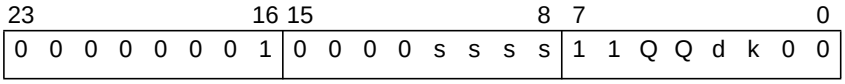
MPY (±)S1,S2,D

MPY (±)S2,S1,D



Instruction Formats and Opcode 2

MPY (±)S,#n,D



MPY(su,uu) Mixed Multiply MPY(su,uu)

Operation

±S1 * S2 → D (S1 unsigned, S2 unsigned)

±S1 * S2 → D (S1 signed, S2 unsigned)

Assembler Syntax

MPYuu (±)S1,S2,D	(no parallel move)
MPYsu (±)S2,S1,D	(no parallel move)

Instruction Fields

{S1,S2}	**QQQQ**	Source registers S1,S2 [all combinations of X0,X1,Y0, and Y1] (see **Table 12-16** on page 12-24)
{D}	**d**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{±}	**k**	Sign [+,–] (see **Table 12-16** on page 12-24)
{s}		[ss,us] (see **Table 12-16** on page 12-24)

Description Multiply the two 24-bit source operands S1 and S2 and store the resulting product in the specified 56-bit destination accumulator D. One or two of the source operands can be unsigned. The "–" sign option is used to negate the specified product prior to accumulation. The default sign option is "+".

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	√	√	√	√	√	√	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and Opcodes

MPY su (±)S1,S2,D

MPY uu (±)S1,S2,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	1	1	s	d	k	Q	Q	Q	Q

MPYI Signed Multiply With Immediate Operand MPYI

Operation

±#xxxxxx*S → D

Assembler Syntax

MPYI (±)#xxxxxx,S,D

Instruction Fields

{S}	**qq**	Source register [X0,Y0,X1,Y1] (see **Table 12-16** on page 12-24)
{D}	**d**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{±}	**k**	Sign [+,–] (see **Table 12-16** on page 12-24)
#xxxx		16-bit Immediate Long Data extension word

Description Multiply the immediate 24-bit source operand #xxxx with the 24-bit register source operand S and store the resulting product in the specified 56-bit destination accumulator D. The "–" sign option is used to negate the specified product prior to accumulation. The default sign option is "+".

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	√	√	√	√	√	√	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and Opcode

MPYI (±)#xxxx,S,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	q	q	d	k	0	0
Immediate Data Extension																							

MPYR Signed Multiply and Round MPYR

Operation		Assembler Syntax	
±S1 * S2 + r → D	(parallel move)	MPYR (±)S1,S2,D	(parallel move)
±S1 * S2 + r → D	(parallel move)	MPYR (±)S2,S1,D	(parallel move)
±(S1 * 2^{-n}) + r → D	(no parallel move)	MPYR (±)S,#n,D	(no parallel move)

Instruction Fields 1

{S1,S2}	**QQQ**	Source registers S1,S2 [X0*X0, Y0*Y0, X1*X0, Y1*Y0, X0*Y1, Y0*X0, X1*Y0, Y1*X1] (see **Table 12-16** on page 12-24)
{D}	**d**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{±}	**k**	Sign [+,–] (see **Table 12-16** on page 12-24)

Instruction Fields 2

{S}	**QQ**	Source register [Y1,X0,Y0,X1] (see **Table 12-16** on page 12-24)
{D}	**d**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{±}	**k**	Sign [+,–] (see **Table 12-16** on page 12-24)
{#n}	**sssss**	Immediate operand (see **Table 12-16** on page 12-24)

Description Multiply the two signed 24-bit source operands S1 and S2 (**or** the signed 16-bit source operand S by the positive 24-bit immediate operand 2^{-n}), round the result using either convergent or two's-complement rounding, and store it in the specified 56-bit destination accumulator D. The "–" sign option negates the product prior to rounding. The default sign option is "+". The contribution of the LS bits of the result is rounded into the upper portion of the destination accumulator. Once the rounding has been completed, the LSBs of the destination accumulator D are loaded with 0s to maintain an unbiased accumulator value that can be reused by the next instruction. The upper portion of the accumulator contains the rounded result that can be read out to the data buses. Refer to the RND instruction for more complete information on the rounding process.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	—
CCR							

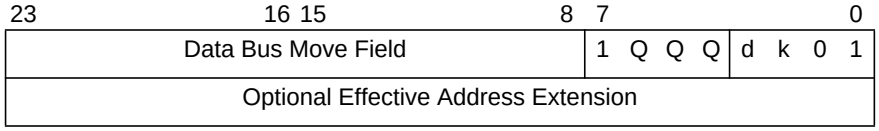
√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and Opcodes 1

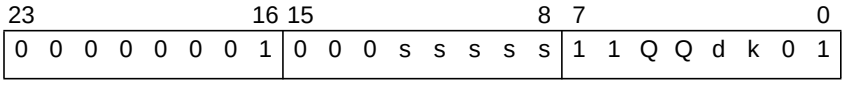
MPYR (±)S1,S2,D

MPYR (±)S2,S1,D



Instruction Formats and Opcode 2

MPYR (±)S,#n,D



Signed Multiply and Round With Immediate Operand

Operation

±#xxxx * S + r → D

Assembler Syntax

MPYRI (±)#xxxx,S,D

Instruction Fields

{S}	**qq**	Source register [X0,Y0,X1,Y1] (see **Table 12-16** on page 12-24)
{D}	**d**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{±}	**k**	Sign [+,–] (see **Table 12-16** on page 12-24)
#xxxx		24-bit Immediate Long Data extension word

Description Multiply the two signed 24-bit source operands #xxxx and S, round the result using either convergent or two's-complement rounding, and store it in the specified 56-bit destination accumulator D. The "–" sign option is used to negate the product before rounding. The default sign option is "+". The contribution of the LS bits of the result is rounded into the upper portion of the destination accumulator. Once the rounding has been completed, the LS bits of the destination accumulator D are loaded with 0s to maintain an unbiased accumulator value that can be reused by the next instruction. The upper portion of the accumulator contains the rounded result that can be read out to the data buses. Refer to the RND instruction for more complete information on the rounding process.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	√	√	√	√	√	√	—
CCR							

√ This bit is changed according to the standard definition.

— This bit is unchanged by the instruction.

Instruction Formats and Opcode

MPYRI (±)#xxxx,S,D

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	q	q	d	k	0	1
Immediate Data Extension																							

NEG — Negate Accumulator — NEG

Operation

0 – D → D (parallel move)

Assembler Syntax

NEG D (parallel move)

Instruction Fields

{D} **d** Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

Description Negate the destination operand D and store the result in the destination accumulator. This is a 56-bit, two's-complement operation.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	—
CCR							

√ Changed according to the standard definition.
— Unchanged by the instruction.

Instruction Formats and Opcodes

NEG D

23 — 16	15 — 8	7 — 0
Data Bus Move Field	0 0 1 1	d 1 1 0
Optional Effective Address Extension		

NOP — No Operation — NOP

Operation

PC+1 → PC

Assembler Syntax

NOP

Instruction Fields

None

Description Increment the Program Counter (PC). Pending pipeline actions, if any, are completed. Execution continues with the instruction following the NOP.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— This bit is unchanged by the instruction.

Instruction Formats and Opcode

	23							16	15							8	7							0
NOP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

NORM Norm Accumulator Iterations NORM

Operation

If $\overline{E} \bullet U \bullet \overline{Z}$=1, then ASL D and Rn–1fiRn
else ifE=1, then ASR D and Rn+1fiR
else NOP

Assembler Syntax

NORM Rn,D

where $\overline{E}$ denotes the logical complement of E and • denotes the logical AND operator

Instruction Fields

{D} **d** Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

{Rn} **RRR** Address register [R0-R7]

Description Perform one normalization iteration on the specified destination operand D, update the specified address register Rn based upon the results of that iteration, and store the result back in the destination accumulator. This is a 56-bit operation. If the accumulator extension is not in use, the accumulator is unnormalized, and the accumulator is not zero, the destination operand is arithmetically shifted one bit to the left, and the specified address register is decremented by 1. If the accumulator extension register is in use, the destination operand is arithmetically shifted one bit to the right, and the specified address register is incremented by 1. If the accumulator is normalized or zero, a NOP is executed and the specified address register is not affected. Since the operation of the NORM instruction depends on the E, U, and Z condition code register bits, these bits must correctly reflect the current state of the destination accumulator prior to executing the NORM instruction.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	√	√	√	√	√	*	—
CCR							

* V Set if bit 55 is changed as a result of a left shift

√ This bit is changed according to the standard definition

— This bit is unchanged by the instruction

Instruction Formats and Opcode

	23							16	15							8	7							0
NORM Rn,D	0	0	0	0	0	0	0	1	1	1	0	1	1	R	R	R	0	0	0	1	d	1	0	1

NORMF Fast Accumulator Normalization NORMF

Operation

If S[23] = 0 then ASR S,D
else ASL -S,D

Assembler Syntax

NORMF S,D

Instruction Fields

{S}	**sss**	Source register [X0,X1,Y0,Y1,A1,B1] (see **Table 12-13** on page 12-22)
{D}	**D**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

Description Arithmetically shift the destination accumulator either left or right as specified by the source operand sign and value. If the source operand is negative then the accumulator is left shifted, and if the source operand is positive then it is right shifted. The source accumulator value should be between +56 to -55 (or +40 to -39 in sixteen bit mode). This instruction can be used to normalize the specified accumulator D, by arithmetically shifting it either left or right so as to bring the leading one or zero to bit location 46. The number of needed shifts is specified by the source operand. This number could be calculated by a previous CLB instruction. For normalization the source accumulator value should be between +8 to -47 (or +8 to -31 in Sixteen- bit Arithmetic mode). NORMF is a 56 bit operation.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	√	√	√	√	√	*	—
CCR							

* V Set if bit 39 is changed any time during the shift operation, and cleared otherwise.
√ Changed according to the standard definition.
— Unchanged by the instruction.

Example

```
CLB     A,B         ;Count leading bits
NORMF   B1,A        ;Normalize A.
```

If the base exponent is stored in R1 it can be updated by the following commands:

```
MOVE    B1,N1       ;Update N1 with shift amount
MOVE    (R1)+N1     ;Increment or decrement exponent
```

Prior to execution, the 56-bit A accumulator contains the value $20:0000:0000. The CLB instruction updates the B accumulator to the number of needed shifts, seven in this example. The NORMF instruction performs seven shifts to the right on A accumulator, and normalization of A is achieved. The exponent register is updated according to the number of shifts.

	Before execution	After execution
`CLB A,B A:`	`$20:0000:0000`	`B: $00:0007:0000`
`NORMF B1,A A:`	`$20:0000:0000`	`A: $00:4000:0000`

Instruction Formats and Opcode

		23 16	15 8	7 0
NORMF	S,D	0 0 0 0 1 1 0 0	0 0 0 1 1 1 1 0	0 0 1 0 s s s D

NOT — Logical Complement — NOT

Operation

$\overline{D[31:16]}$ ⇨ D[31:16] (parallel move)

Assembler Syntax

NOT D (parallel move)

where "—" denotes the logical NOT operator.

Instruction Fields

{D} **d** Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

Description

Take the one's complement of bits 47–24 of the destination operand D and store the result back in bits 47–24 of the destination accumulator. This is a 24-bit operation. The remaining bits of D are not affected.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	*	*	*	—
CCR							

* N Set if bit 47 of the result is set.

* Z Set if bits 47–24 of the result are 0.

* V Always cleared.

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and Opcodes

NOT D

23 — 16	15 — 8	7 — 0
Data Bus Move Field		0 0 0 1 d 1 1 1
Optional Effective Address Extension		

OR — Logical Inclusive OR — OR

Operation

S ⊕ D[47:24] → D[47:24] (parallel move)

#xx ⊕ D[47:24] → D[47:24]

#xxxx ⊕ D[47:24] → D[47:24]

Assembler Syntax

OR S,D (parallel move)

OR #xx,D

OR #xxxx,D

where ⊕ denotes the logical inclusive OR operator.

Instruction Fields

{S}	**JJ**	Source input register [X0,X1,Y0,Y1] (see **Table 12-13** on page 12-22)
{D}	**d**	Destination accumulator [A/B] (see **Table 12-13** on page 12-22)
{#xx}	**iiiiii**	6-bit Immediate Short Data
{#xxxx}		24-bit Immediate Long Data extension word

Description Logically inclusive OR the source operand S with bits 47–24 of the destination operand D and store the result in bits 47–24 of the destination accumulator. The source can be a 24-bit register, 6-bit short immediate, or 24-bit long immediate. This instruction is a 24-bit operation. The remaining bits of the destination operand D are not affected. When using 6-bit immediate data, the data is interpreted as an unsigned integer. That is, the six bits are right aligned, and the remaining bits are zeroed to form a 16-bit source operand.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	*	*	*	—
CCR							

*	N	Set if bit 47 of the result is set.
*	Z	Set if bits 47–24 of the result are 0.
*	V	Always cleared.
√		Changed according to the standard definition.
—		Unchanged by the instruction.

Instruction Formats and Opcodes

OR S,D

23 … 16	15 … 8	7 … 0
Data Bus Move Field		0 1 J J d 0 1 0
Optional Effective Address Extension		

OR #xx,D

23 … 16	15 … 8	7 … 0
0 0 0 0 0 0 0 1	0 1 i i i i i i	1 0 0 0 d 0 1 0

OR #xxxx,D

23 … 16	15 … 8	7 … 0
0 0 0 0 0 0 0 1	0 1 0 0 0 0 0 0	1 1 0 0 d 0 1 0
Immediate Data Extension		

Operation

#xx + D → D

Assembler Syntax

OR(I) #xx,D

where + denotes the logical inclusive OR operator.

Instruction Fields

{D} **EE** Program Controller register [MR,CCR,COM,EOM] (see **Table 12-13** on page 12-22)

{#xx} **iiiiiiii** Immediate Short Data

Description Logically OR the 8-bit immediate operand (#xx) with the contents of the destination control register D and store the result in the destination control register. The condition codes are affected only when the Condition Code Register (CCR) is specified as the destination operand.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
*	*	*	*	*	*	*	*
CCR							

For CCR Operand:

* S Set if bit 7 of the immediate operand is set.

* L Set if bit 6 of the immediate operand is set.

* E Set if bit 5 of the immediate operand is set.

* U Set if bit 4 of the immediate operand is set.

* N Set if bit 3 of the immediate operand is set.

* Z Set if bit 2 of the immediate operand is set.

* V Set if bit 1 of the immediate operand is set.

* C Set if bit 0 of the immediate operand is set.

For MR and OMR Operands:
The condition codes are not affected using these operands.

Instruction Formats and Opcodes

	23 16	15 8	7 0
OR(I) #xx,D	0 0 0 0 0 0 0 0	i i i i i i i i	1 1 1 1 1 0 E E

Operation

Flush instruction cache

Assembler Syntax

PFLUSH

Instruction Fields

None

Description Flush the whole instruction cache, unlock all cache sectors, set the LRU stack and tag registers to their default values. The PFLUSH instruction is enabled only in Cache Mode. When the cache is disabled, execution of this instruction causes an illegal instruction trap.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— This bit is unchanged by the instruction

Instruction Formats and Opcode

	23								16	15							8	7							0
PFLUSH	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Program Cache Flush Unlocked Sections

Operation

Flush Unlocked instruction cache sectors

Assembler Syntax

PFLUSHUN

Instruction Fields

None

Description Flush the instruction cache sectors that are unlocked, set the LRU stack to its default value and set the unlocked tag registers to their default values. The PFLUSHUN instruction is enabled only in Cache mode. When the cache is disabled, execution of this instruction causes an illegal instruction trap.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— This bit is unchanged by the instruction

Instruction Formats and Opcode

	23								16	15							8	7							0
PFLUSHUN	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

PFREE Program Cache Global Unlock PFREE

Operation

Unlock all locked sectors

Assembler Syntax

PFREE

Instruction Fields

None

Description Unlock all the locked cache sectors in the instruction cache. The PFREE instruction is enabled only in Cache Mode. When the cache is disabled, execution of this instruction causes an illegal instruction trap.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction

Instruction Formats and Opcode

	23							16	15							8	7							0
PFREE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Lock Instruction Cache Sector

Operation

Lock sector by effective address

Assembler Syntax

PLOCK ea

Instruction Fields

{ea} **MMMRRR** Effective Address (see **Table 12-13** on page 12-22)

Description Lock the cache sector to which the specified effective address belongs. If the specified effective address does not belong to any cache sector and is therefore definitely locked, nevertheless, load the least recently used cache sector tag with the17 most significant bits of the specified address. Update the LRU stack accordingly. All memory alterable addressing modes can be used for the effective address, but not a short absolute address. The PLOCK instruction is enabled only in Cache mode. In PRAM mode it causes an illegal instruction trap.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction

Instruction Formats and Opcodes

PUNLOCK ea

23								16	15							8	7							0
0	0	0	0	1	0	1	1		1	1	M	M	M	R	R	R	1	0	0	0	0	0	0	1
Address Extension Word																								

Lock Instruction Cache Relative Sector

Operation

Lock sector by PC+xxxx

Assembler Syntax

PLOCKR xxxx

Instruction Fields

None

Description Lock the cache sector to which the sum PC + specified displacement belongs. If the sum does not belong to any cache sector, then load the 17 most significant bits of the sum into the least recently used cache sector tag, and then lock that cache sector. Update the LRU stack accordingly. The displacement is a twos-complement 24-bit integer that represents the relative distance from the current PC to the address to be locked. The PLOCKR instruction is enabled only in Cache Mode. When the cache is disabled, execution of this instruction causes an illegal instruction trap.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction

Instruction Formats and Opcodes

PLOCKR xxxx

23								16	15							8	7							0
0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
ADDRESS EXTENSION WORD																								

Unlock Instruction Cache Sector

Operation

Unlock sector by effective address

Assembler Syntax

PUNLOCK ea

Instruction Fields

{ea} **MMMRRR** Effective Address (see **Table 12-13** on page 12-22)

Description Unlock the cache sector to which the specified effective address belongs. If the specified effective address does not belong to any cache sector, and is therefore definitely unlocked, nevertheless, load the least recently used cache sector tag with the17 most significant bits of the specified address. Update the LRU stack accordingly. All memory alterable addressing modes may be used for the effective address, but not a short absolute address. The PUNLOCK instruction is enabled only in Cache mode. In PRAM mode it causes an illegal instruction trap.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction

Instruction Formats and Opcodes

PUNLOCK ea

23								16	15							8	7							0
0	0	0	0	1	0	1	0		1	1	M	M	M	R	R	R	1	0	0	0	0	0	0	1
Address Extension Word																								

Unlock Instruction Cache Relative Sector

Operation

Unlock sector by PC+xxxx

Assembler Syntax

PUNLOCKR xxxx

Instruction Fields

None

Description Unlock the cache sector to which the sum PC + specified displacement belongs. If the sum does not belong to any cache sector, and is therefore definitely unlocked, nevertheless, load the least recently used cache sector tag with the 17 most significant bits of the sum. Update the LRU stack accordingly. The displacement is a twos-complement 24-bit integer that represents the relative distance from the current PC to the address to be locked. The PUNLOCKR instruction is enabled only in Cache mode. In PRAM mode it causes an illegal instruction trap.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction

Instruction Formats and Opcodes

PUNLOCKR xxxx

23								16	15							8	7							0
0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
Address Extension Word																								

REP — Repeat Next Instruction — REP

Operation

LC → TEMP; [X or y]:ea → LC
Repeat next instruction until LC = 1
TEMP → LC

LC → TEMP; [X or Y]:aa → LC
Repeat next instruction until LC = 1
TEMP → LC

LC → TEMP;S → LC
Repeat next instruction until LC = 1
TEMP → LC

LC → TEMP;#xxx → LC
Repeat next instruction until LC = 1
TEMP → LC

Assembler Syntax

REP [X or Y]:ea

REP [X or Y]:aa

REP S

REP #xxx

Instruction Fields

{ea}	**MMMRRR**	Effective Address
{X/Y}	**S**	Memory Space [X,Y]
{aa}	**aaaaaa**	Absolute Short Address
{#xxx}	**hhhhiiiiiiii**	Immediate Short Data
{S}	**dddddd**	Source register [all on-chip registers]

See **Table 12-13** on page 12-22

Description Repeat the single-word instruction immediately following the REP instruction the specified number of times. The value specifying the number of times the given instruction is to be repeated is loaded into the 24-bit loop counter (LC) register. The single-word instruction is then executed the specified number of times, decrementing the loop counter (LC) after each execution until LC = 1. When the REP instruction is in effect, the repeated instruction is fetched only one time, and it remains in the instruction register for the duration of the loop count. Thus, the REP instruction is not interruptible (sequential repeats are also not interruptible). The current loop counter (LC) value is stored in an internal temporary register. If LC is set equal to zero, the instruction is repeated 65,536 times. The instruction's effective address specifies the address of the value which is to be loaded into the loop counter (LC). All address register indirect addressing modes can be used. The absolute short and the immediate short addressing modes may also be used. The four MS bits of the 12-bit immediate value are zeroed to form the 24-bit value that is to be loaded into the loop counter (LC).

If the System Stack register SSH is specified as a source operand, the system Stack Pointer (SP) is post-decremented by 1 after SSH has been read.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and Opcodes

		23 16	15 8	7 0
REP	[X or Y]:ea	0 0 0 0 0 1 1 0	0 1 M M M R R R	0 S 1 0 0 0 0 0
REP	[X or Y]:aa	0 0 0 0 0 1 1 0	0 0 a a a a a a	0 S 1 0 0 0 0 0
REP	#xxx	0 0 0 0 0 1 1 0	i i i i i i i i	1 0 1 0 h h h h
REP	S	0 0 0 0 0 1 1 0	1 1 d d d d d d	0 0 1 0 0 0 0 0

Operation

Reset the interrupt priority register and all on-chip peripherals

Assembler Syntax

RESET

Instruction Fields

None.

Description Reset the interrupt priority register and all on-chip peripherals. This is a software reset, which is *not* equivalent to a hardware $\overline{\text{RESET}}$ since only on-chip peripherals and the interrupt structure are affected. The processor state is not affected, and execution continues with the next instruction. All interrupt sources are disabled except for the stack error, NMI, illegal instruction, Trap, Debug request, and hardware reset interrupts.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and Opcode

	23 16	15 8	7 0
RESET	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 0 0 0 0 1 0 0

RND Round Accumulator RND

Operation

D + r → D (parallel move)

Assembler Syntax

RND D (parallel move)

Instruction Fields

{D} d Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

Description Round the 56-bit value in the specified destination operand D and store the result in the destination accumulator (A or B). The contribution of the LSBs of the operand is rounded into the upper portion of the operand by adding a rounding constant to the LSBs of the operand. The upper portion of the destination accumulator contains the rounded result. The boundary between the lower portion and the upper portion is determined by the scaling mode bits S0 and S1 in the Status Register (SR).

Two types of rounding can be used: convergent rounding (also called round to nearest (even)) or twos-complement rounding. The type of rounding is selected by the Rounding Mode bit (RM) in the MR portion of the SR. In both rounding modes a rounding constant is first added to the unrounded result. The value of the rounding constant added is determined by the scaling mode bits S0 and S1 in the SR. A 1 is positioned in the rounding constant aligned with the MSB of the current LS portion, that is, the rounding constant weight is actually equal to half the weight of the upper portion's LSB. The following table shows the rounding position and rounding constant as determined by the scaling mode bits:

			Rounding	Rounding Constant				
S1	**S0**	**Scaling Mode**	**Position**	**55 - 25**	**24**	**23**	**22**	**21 - 0**
0	0	No Scaling	23	0. . . .0	0	1	0	0. . . .0
0	1	Scale Down	24	0. . . .0	1	0	0	0. . . .0
1	0	Scale Up	22	0. . . .0	0	0	1	0. . . .0

If convergent rounding is used, the result of this addition is tested and if all the bits of the result to the right of, and including, the rounding position are cleared, then the bit to the left of the rounding position is cleared in the result. This ensures that the result is not biased. In both rounding modes, the Least Significant Bits (LSBs) of the result are cleared. The number of LSBs cleared is determined by the Scaling Mode bits in the Status Register (SR). All bits to the right of and including the rounding position are cleared in the result.

In Sixteen-bit Arithmetic mode the 40-bit value (in the 56-bit destination operand D) is rounded and stored in the destination accumulator (A or B). This implies that the

boundary between the lower portion and upper portion is in a different position then in 24 bit mode. The following table shows the rounding position and rounding constant in sixteen bit arithmetic mode, as determined by the scaling mode bits:

			Rounding	Rounding Constant				
S1	**S0**	**Scaling Mode**	**Position**	**55 - 33**	**32**	**23**	**22**	**21 - 8**
0	0	No Scaling	31	0. . . .0	0	1	0	0. . . .0
0	1	Scale Down	32	0. . . .0	1	0	0	0. . . .0
1	0	Scale Up	30	0. . . .0	0	0	1	0. . . .0

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	—
CCR							

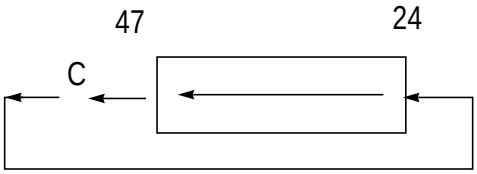
√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and Opcodes

		23 16	15 8	7 0
RND	D	Data Bus Move Field	0 0 0 1	d 0 0 1
		Optional Effective Address Extension		

Operation



Assembler Syntax

ROL D (parallel move)

Instruction Fields

{D} **d** Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

Description Rotate bits 47–24 of the destination operand D one bit to the left and store the result in the destination accumulator.The Carry bit (C) receives the previous value of bit 47 of the operand.The previous value of the C bit is shifted into bit 24 of the operand.This instruction is a 24-bit operation. The remaining bits of destination operand D are not affected.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	*	*	*	*
CCR							

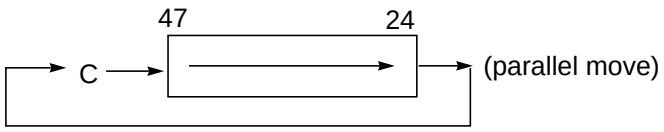
* N Set if bit 47 of the result is set.
* Z Set if bits 47–24 of the result are 0.
* V This bit is always cleared.
* C Set if bit 47 of the destination operand is set, and cleared otherwise.
√ Changed according to the standard definition.
— Unchanged by the instruction.

Instruction Formats and Opcodes

ROL D

23 — 16	15 — 8	7 — 0
Data Bus Move Field		0 0 1 1 d 1 1 1
Optional Effective Address Extension		

Operation



Assembler Syntax

ROR D (parallel move)

Instruction Fields

{D} **d** Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

Description Rotate bits 47–24 of the destination operand D one bit to the right and store the result in the destination accumulator.The Carry bit (C) receives the previous value of bit 24 of the operand.The previous value of the C bit is shifted into bit 47 of the operand. This instruction is a 24-bit operation. The remaining bits of destination operand D are not affected.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	*	*	*	*
CCR							

* N Set if bit 47 of the result is set.
* Z Set if bits 47–24 of the result are 0.
* V Always cleared.
* C Set if bit 47 of the destination operand is set, and cleared otherwise.
√ Changed according to the standard definition.
— Unchanged by the instruction.

Instruction Formats and Opcodes

		23 16	15 8	7 0
ROR	D	Data Bus Move Field	0 0 1 0	d 1 1 1
		Optional Effective Address Extension		

RTI — Return From Interrupt — RTI

Operation

SSH → PC; SSL → SR; SP – 1 → SP

Assembler Syntax

RTI

Instruction Fields

None.

Description Pull the Program Counter (PC) and the Status Register (SR) from the system stack. The previous PC and SR values are lost.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
*	*	*	*	*	*	*	*
CCR							

* S Set according to the value pulled from the stack.
* L Set according to the value pulled from the stack.
* E Set according to the value pulled from the stack.
* U Set according to the value pulled from the stack.
* N Set according to the value pulled from the stack.
* Z Set according to the value pulled from the stack.
* V Set according to the value pulled from the stack.
* C Set according to the value pulled from the stack.

Instruction Formats and Opcode

	23 16	15 8	7 0
RTI	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0

RTS — Return From Subroutine — RTS

Operation

SSH → PC; SP – 1 → SP

Assembler Syntax

RTS

Instruction Fields

None.

Description Pull the Program Counter (PC) from the system stack. The previous PC value is lost. The Status Register (SR) is not affected.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and Opcode

	23							16	15							8	7							0
RTS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0

SBC — Subtract Long With Carry — SBC

Operation

D – S – C → D (parallel move)

Assembler Syntax

SBC S,D (parallel move)

Instruction Fields

{S}	**J**	Source register [X,Y] (see **Table 12-13** on page 12-22)
{D}	**d**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

Description Subtract the source operand S and the Carry bit(C) from the destination operand D and store the result in the destination accumulator. Long words (48-bit words) are subtracted from the 56-bit destination accumulator. Note that the C bit is set correctly for multiple-precision arithmetic using long-word operands if the extension register of the destination accumulator (A2 or B2) is the sign extension of bit 47 of the destination accumulator (A or B).

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	√
CCR							

√ Changed according to the standard definition.

Instruction Formats and Opcodes

SBC S,D

23 — 16	15 — 8	7 — 0
Data Bus Move Field		0 0 1 J d 1 0 1
Optional Effective Address Extension		

STOP Stop Instruction Processing STOP

Operation

Enter the stop processing state and stop the clock oscillator

Assembler Syntax

STOP

Instruction Fields

None

Description Enter the Stop processing state. All activity in the processor is suspended until the $\overline{\text{RESET}}$ or $\overline{\text{IRQA}}$ pin is asserted or the Debug Request JTAG command is detected. The clock oscillator is gated off internally. The Stop processing state is a low-power standby state. During the Stop state, the destination port is in an idle state with the control signals held inactive, the data pins are high impedance, and the address pins are unchanged from the previous instruction. If the exit from the Stop state is caused by a low level on the $\overline{\text{RESET}}$ pin, then the processor enters the reset processing state. If the exit from the Stop state was caused by a low level on the $\overline{\text{IRQA}}$ pin, then the processor will service the highest priority pending interrupt and will not service the $\overline{\text{IRQA}}$ interrupt unless it is highest priority. If no interrupt is pending, the processor will resume program execution at the instruction following the STOP instruction that caused the entry into the Stop state. Program execution (interrupt or normal flow) resumes after an internal delay counter counts:

- If the Stop Delay (SD, OMR[6]) bit is cleared—131,070 clock cycles
- If the Stop Delay (SD, OMR[6]) bit is set—24 clock cycles
- If the Stop Processing State (PSTP, PCTL1[5]) is set—8.5 clock cycles

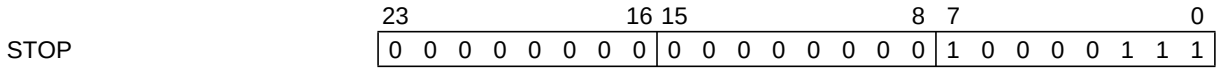
During the clock stabilization count delay, all peripherals and external interrupts are cleared and re-enabled/arbitrated at the end of the count interval. If the $\overline{\text{IRQA}}$ pin is asserted when the STOP instruction is executed, the clock is not gated off, and only the internal delay counter is started.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and Opcode



SUB — Subtract — SUB

Operation

D–S → D	(parallel move)
D – #xx → D	
D – #xxxx → D	

Assembler Syntax

SUB S, D	(parallel move)
SUB #xx, D	
SUB #xxxx,D	

Instruction Fields

{S}	**JJJ**	Source register [B/A,X,Y,X0,Y0,X1,Y1] (see **Table 12-13** on page 12-22)
{D}	**d**	Destination accumulator [A/B] (see **Table 12-13** on page 12-22)
{#xx}	**iiiiii**	6-bit Immediate Short Data
{#xxxx}		24-bit Immediate Long Data extension word

Description Subtract the source operand from the destination operand D and store the result in the destination operand D. The source can be a register (24-bit word, 48-bit long word, or 56-bit accumulator), 6-bit short immediate, or 24-bit long immediate. When using 6-bit immediate data, the data is interpreted as an unsigned integer. That is, the six bits are right-aligned and the remaining bits are zeroed to form a 16-bit source operand. Note that the Carry bit (C) is set correctly using word or long-word source operands if the extension register of the destination accumulator (A2 or B2) is the sign extension of bit 47 of the destination accumulator (A or B). The C bit is always set correctly using accumulator source operands.

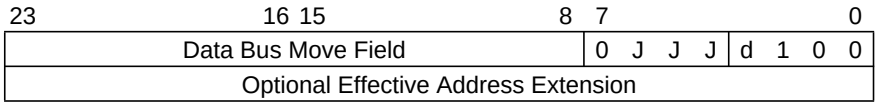
Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	√
CCR							

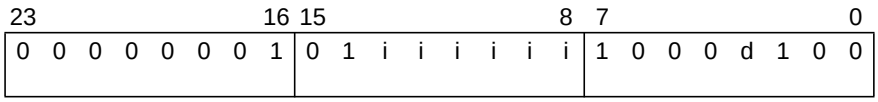
√ Changed according to the standard definition.

Instruction Formats and Opcodes

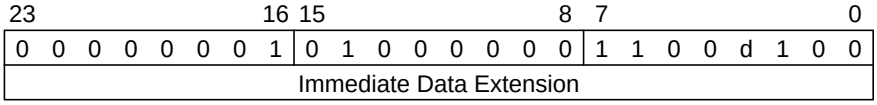
SUB S,D



SUB #xx,D



SUB #xxxx,D



SUBL Shift Left and Subtract Accumulators SUBL

Operation

$2 * D - S \rightarrow D$ (parallel move)

Assembler Syntax

SUBL S,D ((parallel move)

Instruction Fields

{D} **d** Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

{S} The source accumulator is B if the destination accumulator (selected by the **d** bit in the opcode) is A, or A if the destination accumulator is B

Description Subtract the source operand S from two times the destination operand D and store the result in the destination accumulator. The destination operand D is arithmetically shifted one bit to the left, and a 0 is shifted into the LSB of D prior to the subtraction operation. The Carry bit (C) is set correctly if the source operand does not overflow as a result of the left shift operation. The Overflow bit (V) may be set as a result of either the shifting or subtraction operation (or both). This instruction is useful for efficient divide and Decimation-In-Time (DIT) FFT algorithms.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	*	√
CCR							

* V Set if overflow has occurred in the result or if the MS bit of the destination operand is changed as a result of the instruction's left shift

√ Changed according to the standard definition

Instruction Formats and Opcodes

SUBL S,D

23 16	15 8	7 0
Data Bus Move Field		0 0 0 1 d 1 1 0
Optional Effective Address Extension		

SUBR Shift Right and Subtract Accumulators SUBR

Operation

D / 2 – S → D (parallel move)

Assembler Syntax

SUBR S,D parallel move)

Instruction Fields

{D} **d** Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

{S} The source accumulator is B if the destination accumulator (selected by the **d** bit in the opcode) is A, or A if the destination accumulator is B

Description Subtract the source operand S from one-half the destination operand D and store the result in the destination accumulator. The destination operand D is arithmetically shifted one bit to the right while the MS bit of D is held constant prior to the subtraction operation. In contrast to the SUBL instruction, the Carry bit (C) is always set correctly, and the Overflow bit (V) can only be set by the subtraction operation, and not by an overflow due to the initial shifting operation. This instruction is useful for efficient divide and Decimation-In-Time (DIT) FFT algorithms.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	√
CCR							

√ Changed according to the standard definition.

Instruction Formats and Opcodes

SUBR S,D

23 16	15 8	7 0
Data Bus Move Field	0 0 0 0	d 1 1 0
Optional Effective Address Extension		

Operation

If cc, then S1 → D1

If cc, then S1 → D1 and S2 → D2

If cc, then S2 → D2

Assembler Syntax

Tcc	S1,D1
Tcc	S1,D1 S2,D2
Tcc	S2,D2

Instruction Fields

{cc}	**CCCC**	Condition code (see **Table 12-16** on page 12-24)
{S1}	**JJJ**	Source register [B/A,X0,Y0,X1,Y1] (see **Table 12-16** on page 12-24)
{D1}	**d**	Destination accumulator [A/B] (see **Table 12-13** on page 12-22)
{S2}	**ttt**	Source address register [R0–R7]
{D2}	**TTT**	Destination Address register [R0–R7]

Description Transfer data from the specified source register S1 to the specified destination accumulator D1 if the specified condition is true. If a second source register S2 and a second destination register D2 are also specified, transfer data from address register S2 to address register D2 if the specified condition is true. If the specified condition is false, a NOP is executed. The conditions that “cc” can specify are listed on **Table 12-16** on page 12-24. When used after the CMP or CMPM instructions, the Tcc instruction can perform many useful functions, such as a “maximum value,” “minimum value,” “maximum absolute value,” or “minimum absolute value” function. The desired value is stored in the destination accumulator D1. If address register S2 is used as an address pointer into an array of data, the address of the desired value is stored in the address register D2. The Tcc instruction may be used after any instruction and allows efficient searching and sorting algorithms. The Tcc instruction uses the internal Data ALU paths and internal Address ALU paths. It does not affect the condition code bits.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and Opcode

Tcc S1,D1

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	C	C	C	C	0	0	0	0	0	J	J	J	d	0	0	0

Tcc S1,D1 S2,D2

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	C	C	C	C	0	t	t	t	0	J	J	J	d	T	T	T

Tcc S2,D2

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	C	C	C	C	1	t	t	t	0	0	0	0	0	T	T	T

TFR — Transfer Data ALU Register — TFR

Operation

S → D (parallel move)

Assembler Syntax

TFR S,D (parallel move)

Instruction Fields

{S} **JJJ** Source register [B/A,X0,Y0,X1,Y1] (see **Table 12-16** on page 12-24)

{D} **d** Destination accumulator [A/B] (see **Table 12-13** on page 12-22)

Description Transfer data from the specified source Data ALU register S to the specified destination Data ALU accumulator D. TFR uses the internal Data ALU data paths; thus, data does not pass through the data shifter/limiters. This allows the full 56-bit contents of one of the accumulators to be transferred into the other accumulator *without* data shifting and/or limiting. Moreover, since TFR uses the internal Data ALU data paths, parallel moves are possible.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and Opcodes

TFR S,D

23 — 16	15 — 8	7 — 0
Data Bus Move Field	0 J J J	d 0 0 1
Optional Effective Address Extension		

Operation

Begin trap exception process

Assembler Syntax

TRAP

Instruction Fields

None

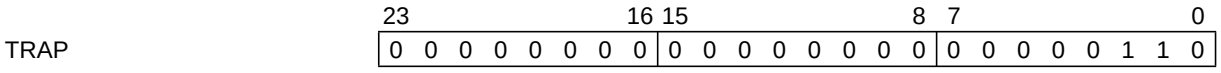
Description Suspend normal instruction execution and begin TRAP exception processing. The Interrupt Priority Level (I1,I0) is set to 3 in the Status Register (SR) if a long interrupt service routine is used.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and Opcode



TRAPcc Conditional Software Interrupt TRAPcc

Operation

If cc then begin software exception processing

Assembler Syntax

TRAPcc

Instruction Fields

{cc} **CCCC** Condition code (see **Table 12-18** on page 12-28)

Description If the specified condition is true, normal instruction execution is suspended and software exception processing is initiated. The Interrupt Priority Level (I1,I0) is set to 3 in the Status Register (SR) if a long interrupt service routine is used. If the specified condition is false, instruction execution continues with the next instruction. The conditions that the term “cc” can specify are listed on **Table 12-18** on page 12-28.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and Opcode

	23							16	15							8	7							0
TRAPcc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	C	C	C	C

Operation

S – 0 (parallel move)

Assembler Syntax

TST S (parallel move)

Instruction Fields

{S} **d** Source accumulator [A,B] (see **Table 12-13** on page 12-22)

Description

Compare the specified source accumulator S with 0 and set the condition codes accordingly. No result is stored although the condition codes are updated.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	*	—
CCR							

* V Always cleared.
√ Changed according to the standard definition.
— Unchanged by the instruction.

Instruction Formats and Opcodes

TST S

23 16	15 8	7							0
Data Bus Move Field		0	0	0	0	d	0	1	1
Optional Effective Address Extension									

Operation

S[47:24] → X:ea; {S[23:0],i} → Y:ea

Assembler Syntax

VSL S,i,L:ea

Instruction Fields

{S}	**S**	Source register A,B (see **Table 12-13** on page 12-22)
{i}	**i**	Bit value, 0 or 1 to be placed in the least significant bit of Y:<ea>
{ea}	**MMMRRR**	Effective address (see **Table 12-13** on page 12-22)

Description Store the most significant part (24 bits) of the source accumulator at X memory (at effective address location), while for the least significant part (24 bits) of the source accumulator shift one bit to the left and insert 0 or 1 at the Least Significant Bit, according to operand i, and store the result at Y memory at the same address. This instruction enhances Viterbi algorithm performance.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and Opcodes

VSL S,i,L:ea

23							16	15							8	7							0
0	0	0	0	1	0	1	S	1	1	M	M	M	R	R	R	1	1	0	i	0	0	0	0
Optional Effective Address Extension																							

WAIT — Wait for Interrupt or DMA Request — WAIT

Operation

Disable clocks to the processor core and enter the Wait processing state

Assembler Syntax

WAIT

Instruction Fields

None

Description Enter the low-power standby Wait processing state. The internal clocks to the processor core and memories are gated off, and all activity in the processor is suspended until an unmasked interrupt occurs. The clock oscillator and the internal I/O peripheral clocks remain active. If the WAIT instruction is executed when an interrupt is pending, the interrupt is processed. The effect is the same as if the processor never entered the Wait state. When an unmasked interrupt or external (hardware) processor reset occurs, the processor leaves the Wait state and begins exception processing of the unmasked interrupt or reset condition. The processor also exits from the Wait state when the Debug Request ($\overline{DE}$) pin is asserted or when a Debug Request JTAG command is detected.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction

Instruction Formats and Opcode

	23								16	15							8	7							0
WAIT	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0